Valerio Colitta
Théo Foray

TDTS06 – Computer Networks

Transmission Control Protocol (TCP)

Assignment 3

Valerio Colitta
Théo Foray

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.
   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.
6. The sequence number of the segment containing the POST is: 232293053.
7. 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

8. Segment1: TCP Length = 565
   Segment2: TCP Length = 1460
   Segment3: TCP Length = 1460
   Segment4: TCP Length = 1460
   Segment5: TCP Length = 1460
   Segment6: TCP Length = 1460

Valerio Colitta
Théo Foray

9. 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.
10. 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.
11. 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.
12. During the TCP connection the throughput is roughly 31KB/s. We calculated it by taking the total number of bytes sent, dividing it with the whole TCP connection time. (the precise calculation is: 164040 /5.3 = 31KB/s)
    164040 (sequence number of the last – sequence number of the first segment)
    5.270854 (time where the last segment is received – time where the first is received)

Paragraph:

At a high level, client and server identify each other and the messages received, by using the standard procedures such as the number of the port for the connection, the sequence numbers, and all the information contained in a segment (eg flags).

TCP Congestion Control in Action:

13. First, the two graphs are very different.
    The second one for example seems to increase the number of sent packets each "interval", and then, after some threshold sends them constantly with the time. The window size at receiver size never decreases, so it's probable that the congestion phase never takes over. Moreover, beside just one DUPACK (in figure 2b), there are no packet retransmission (since the number of the sequence number is always strictly monotonically increasing). The first one acts similarly (no congestion control), but the way of sending packets is different. This may be application dependent.

14. 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 if 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 minimum value between (RWND and CWND).

15. 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.

    Paragraph:

    By thaking a look at both these traces, the congestion control never takes over. We can safely state that by noticing that since there are no retrasmissions (by timeout or dupacks), then the network is not overloaded. Moreover, the receiver windows never decreases, which implies that no packets at the receiver side are lost (it can process them in time). Finally since the sender effective window always increases, then it mean that the CWND is the one that is increasing (Effective window increases over time, Effective window = min(CWND, RWND), and RWND is fixed, ⇒CWND increases ⇒CWND doesn't decrease ⇒no congestion control mechanism activated).

A Short Study of TCP Fairness:

16. 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.

17.

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 reasons of the low throughput are: because of the Round-Trip Time but mostly because, as it is said, the servers are from all around the world so it could imply different bottleneck routers that will change the throughput.

18. In the BiTorrent case, the TCP Fairness is a bit 'altered'. We see that the Total bytes transferred, the Durantion and the RTT is different and nothing really looks linear between the connections. The explanation is because BitTorrent Protocol is, sort of, breaking TCP Fairness. In fact, in the end it is helping the whole network, because, yes at the beginning the hosts with the most bandwidth and who are the more active on the network will get chunks faster, but quickly all the others will benefit from it (so it restores fairness, but another way) because more chunks will be available and on different hosts.