LINKÖPINGS UNIVERSITET

TDTS04 Computer Networks and Distributed Systems

Assignment 3: Transport-layer and TCP friendly protocols

Author

Christer Vesterlund, chrve180@student.liu.se
Denis I. Blazevic denbl369@student.liu.se

INSTITUTIONEN FÖR DATAVETENSKAP

2017-02-16

Innehåll

1 Tasks

This assignment is split into three parts and tries to cover TCP in general.

1.1 Task A

This is what we concluded from the questions 1-12, that covers TCP basic communications.

1.1.1 TCP connections at a high level

When a client wants to established a connection with another client (or server) and uses TCP protocol, the first thing thats required is a handshake. To accomplish that the client sends a SYN-segment to the receiver/server asking to set up a connection. If the server accepts this incoming request, the server responds with a SYNACK-segment to confirm its willingness to set up a communication. Finally the client confirm this with a ACK back to the server. One thing worth mentioning is that in the first ACK sent by the client there was "Sequence number: 0" included. This sequence number is included on each transmitted packet, and acknowledged by the server as an acknowledgment number to inform the sending part that the transmitted data was received successfully.

Now we have a open connection. In this assignment the client starts by sending a HTTP POST request. When the request has been transmitted the server respond with an ACK which includes the "Sequence number", "Next sequence number" and "Acknowledgment number". The Acknowledgment number tells the client that the server have received this many bytes so far in this segment. The Next sequence tells the client which packages the server is expecting next.

Over time the server tries to optimize the connection by not ACK every segment, but instead every other received segment.

To prevent the server to be over-flooded, there are ways to control the congestion. In the TCP protocol the server put information in the ACK how much space it have left in its buffer. The client uses this to control how much to send or to slow down the flow of packages.

1.1.2 RTT estimates

The RTT stands for round trip time and is the time for a message sent to be sent and the ACK is received The only way for the client to know if a packet was lost in the transfer is to make a estimation of the time to get the ACK. If the ACK hasn't arrived in that time, the client makes an educated guess that the package was lost. When this happens the client tries to send it again. The general recommendation is that the estimated RTT should be larger then the RTT plus a small margin, but this margin depends on the RTT deviation. If this deviation has a big fluctuation, the margin must increase proportionally to the deviation.

1.2 Task B

1.2.1 Question 13.

Identify where TCP's slow start phase begins and ends.

The first file (*tcp-ethereal-trace-1*) Here we can se that the slow-start begins at package 4 and ends at package 13. So the slow-start last for about 0.15 seconds. How can we see this? Well if we look at the sequence graph (Stevens), so can we clearly see a curve similar to an quadratic equation. After that time mark the following packages are sent in a linear rate, with 5 package in each burst. Why does the client do this? Well we cant find any indication of congestion avoidance, so our guess is that the client never send out enough data to make the congestion avoidance to become active.

The second file (*filtered_573_packets_anonymized.pcap*)

With the same technique as the previous file, we examined the graph. We found that slow-start begins at packages 11 and continued until package 35. After that the congestion control starts. We can see this in the graph, because after the slow-start phase it enters a congestion avoidance state after reaching the same amount of packets sent as the last batch of packets. This tells us that the send rate changes from quadratic equation growth to linear growth (congestion avoidance).

1.2.2 Question 14.

Explain the relationship between (i) the congestion window, (ii) the receiver advertised window, (iii) the number of unacknowledged bytes, and (iv) the effective window at the sender.

1. Congestion window
   This is a sender imposed window that was implemented to avoid over-flooding routers. The sender, after each segment sent, will increase its congestion window slightly, i.e. the sender will allow itself to send more data. But if the sender detects packet loss, it will cut the window into half to keep less data in flight. Once the window reaches the Slow Start threshold, or there is data loss, the growth of the window will change to a special congestion avoidance algorithm.

2. Receiver advertised window
   The receive window is managed by the receiver, who sends out window sizes to the sender. The window sizes tells the sender how many bytes are still free in the receiver buffer, i.e. how many bytes the sender can send to the receiver without needing an acknowledgment.

3. Unacknowledged bytes
   This is basically the bytes already sent to the receiver, i.e. bytes "in flight".

4. Effective sender window
   This is basically the minimum among the receiver advertised window and the congestion window at a specific time.

To sum it up, these factors will control the sender's throughput. The sender will not be able to send more data than the advertised receiver window, nor will the sender be able to send more data than the congestion window.

1.2.3 Question 15.

Is it generally possible to find the congestion window size (i.e. cwnd).

It is not possible to find the congestion window because, as defined in RFC 2581, it is not advertised. It is, however, possible to "estimate" the congestion window at any time, if you know the senders congestion algorithm and the configuration values, by looking at the sent frames and the received ACKs. As far as we know, this is not possible in Wireshark.

1.3 Task C

1.3.1 Question 16.

What is the throughput of each of the connections in bps (bits per second)? What is the total bandwidth of the host on which the clients are running? Discuss the TCP fairness for this case.

Connection	Total transferred bytes	Duration (in seconds)	RTT (in milliseconds)	BPS
1	165,095,720	521	12	316,882
2	165,842,766	521	12	318,316
3	165,458,792	514	12	321,904
4	163,235,772	512	12	318,819

Tabell 1: Concurrent downloads

To calculate the BPS, we simply took TTB divided by the duration. This tells us that the total bandwith is 1,275,921 bytes with these four connections. In this case the fairness would come to 318,980 bps for each connection, and the biggest deviation comes to ~5000 bytes. So our conclution are that this is fair.

1.3.2 Question 17.

What is the throughput of each of the connections in bps (bits per second)? What is the total bandwidth of the host on which the clients are running? Discuss the TCP fairness for this case.

Connection	Total transferred bytes	Duration (in seconds)	RTT (in milliseconds)	BPS
1	261,319,130	90	13	2,903,545
2	175,995,832	90	35	1,955,509
3	151,894,552	90	68	1,687,717
4	140,388,568	90	73	1,559,872
5	108,610,702	90	49	1,206,785
6	70,644,690	90	33	784,941
7	65,744,938	90	135	730,499
8	43,212,876	90	326	480,143
9	39,222,524	90	322	435,805

Tabell 2: Concurrent downloads

The total bandwidth of these connection is 11,844,816 bytes for nine connections.
Connection 1 takes majority of the total bandwidth because it has the fastest RTT, which makes it the fastest responding connection in this scenario.
Connections 2 and 6 comes second in place in terms of fast responding connections. This is because the RTT is almost the same between them, which makes them respond almost at the same time. However connection

6 will be given less BPS because it has less bytes to transfer.
Connection 3-5 will be given less fairness but still be some of the highest BPS connections because the TTB vs RTT is reasonable for the scenario.
Connections 7-9 will be the worst case of fairness in this scenario. The RTT is very large in comparison to the other connections, this will most likely cause packet loss because the other connections can cause a blockage for these connections.
We would say this is TCP fair, because the connections 1-6 has much lower RTT than connections 7-9 and should therefore claim the bandwidth. Otherwise the connections 7-9 would more or less block the rest of the connections from completing the transfers.

1.3.3 Question 18.

Discuss the TCP fairness for this case.

Connection	Total transferred bytes	Duration (in seconds)	RTT (in milliseconds)	BPS
1	108,851,134	58	40	1,876,744
2	90,435,681	58	36	1,559,236
3	57,971,584	53	100	1,093,803
4	32,000,012	29	68	1,103,449
5	32,557,334	35	31	930,210
6	27,199,361	31	33	877,399
7	26,329,578	31	122	849,341
8	38,834,490	56	146	693,473
9	23,571,761	35	74	673,478
10	36,252,962	55	66	659,145

Tabell 3: Concurrent downloads

This table is an BitTorrent example where we again have different download sources in the form of peers/users. The paths between the client and the peers should differ and so will the bottlenecks (if they occur). The sending rate of the seeders will not always be uniform. The P2P client (at least on the application layer) can decide to prioritize the upload/download from specific peers with faster possible throughput, this will be affected by the clients ability to upload data back. Total transferred bytes also include 'uploaded bytes' in this table.