Computer Communications and Networks (COMN) 2021/22, Semester 2

Assignment 2 Results Sheet

Forename and Surname:	Kenza Amira
Matriculation Number:	s1813674

Question 1 – Number of retransmissions and throughput with different retransmission timeout values with stop-and-wait protocol. For each value of retransmission timeout, run the experiments for **5 times** and write down **average number of retransmissions** and **average throughput**.

Retransmission timeout (ms)	Average number of re-transmissions	Average throughput (Kilobytes per second)
5	978.40	69.008
10	629.40	60.536
15	123.00	60.427
20	126.60	52.246
25	108.40	60.412
30	111.40	50.654
40	105.60	44.151
50	106.20	42.375
75	117.20	31.448
100	94.20	36.0995

Question 2 – Discuss the impact of retransmission timeout value on the number of retransmissions and throughput. Indicate the optimal timeout value from a communication efficiency viewpoint (i.e., the timeout that minimizes the number of retransmissions while ensuring a high throughput).

When the retransmission timeout is lower or equal to 10ms, we experience a high number of re-transmision with high throughput. This is due to the fact that the average RTT is 10ms and thus acks are sent 10ms after the receipt of a packet. The ack doesn't have enough time to be sent and so the packets are retransmitted a high number of times. This also seems to be applying to the 15/20ms timeout although to a lesser extent.

As timeouts increase, we observe a lower number of re-transmissions but also a lower average throughput. In fact, having a higher timeout only causes unnecessary delay as the ack won't be received in that period of time and the packet will have to be retransmitted anyway. However, we will still have to wait for the timeout to occur before retransmitting. This is even more observable if we look at the ratio

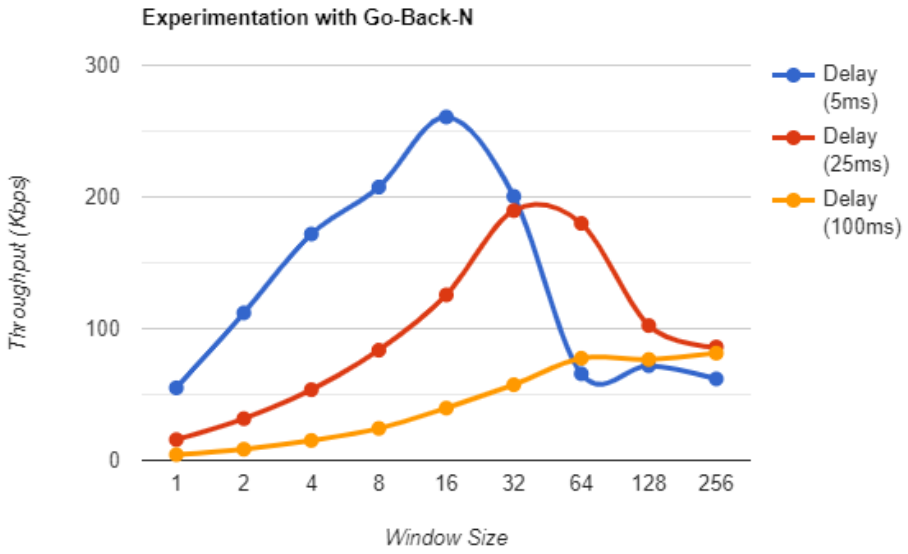
number of retransmission/ total packets(which in our case for the test file is 879 packets.), as the more the timeout increases, the closer the value gets to the packet loss (5%).

From the above experiments, it appears that the optimal value is **25ms** as it has a somewhat low number of re-transmissions while keeping high average throughput.

Question 3 – Experimentation with Go-Back-N. For each value of window size, run the experiments for **5 times** and write down **average throughput**.

	Average throughput (Kilobytes per second)		
Window Size	**Delay = 5ms**	**Delay = 25ms**	**Delay = 100ms**
1	55.067	15.743	4.329
2	111.967	31.636	8.460
4	171.850	53.668	15.105
8	207.506	83.635	24.234
16	260.721	125.632	39.684
32	200.486	132.637	57.431
64	65.762	119.966	77.521
128	71.888	102.360	76.596
256	61.881	86.051	81.405

Create a graph as shown below using the results from the above table:



Question 4 – Discuss your results from Question 3.

For this task, I've used the optimal timeout value defined as part of Question 2 for the 5ms timeout. I've then used a 65ms and 215ms timeout for the 25ms and 100ms delays, respectively. The reason why I opted for these values follows the reasoning used to answer Q2. As the average time to send an ack is defined by 2*one way delay. I pick values greater than (and within reasonable range from) 50 (25 * 2) and 200 (100 * 2).

From the above results, an interesting observation is that, as we increase the window size, the throughput also increases. This is mainly due to the fact that we are sending out bigger chunks. However, when the window size gets too large (128 and 256 mostly), the throughput starts decreasing again. An explanation for that is that the Receiver isn't able to send acknowledgments in time as they receive too many packets which in turn makes it harder to move the window forward.

This also varies depending on the delay used. As seen in the table, the 5ms delay optimal window size is 16 after which the values start decreasing. For the 25ms delay it is 32 and for the 100ms delay it is 256. The reason behind that is that the higher the delay, the slower the packets are being sent. This means that the receiver struggles less to send acks on time as it has more time to do so.

However, while the window size might be larger, the value for which the throughput peaks decreases as we increase the delay. In fact, the Go-back-N protocol re-transmits all the packets in the window even if the issue only comes from one. So, as the window size increases, we are forced to re-transmit more packets should there be an issue.

Question 5 – Experimentation with Selective Repeat. For each value of window size, run the experiments for **5 times** and write down **average throughput**.

	Average throughput (Kilobytes per second)
Window	**Delay = 25ms**
1	11.081
2	17.737
4	47.564
8	65.501
16	130.743
32	149.919

Question 6 - Compare the throughput obtained when using "Selective Repeat" with the corresponding results you got from the "Go Back N" experiment and explain the reasons behind any differences.

For this experiment, I've set the timeout at 65ms as before to have a good comparison point.

On smaller window sizes, we don't observe much of a difference. This makes sense as sending the whole window would take approximately just as much time as sending single packets. However, as window size increases, we obtain a higher throughput overall for Selective Repeat.

This is to be expected as Selective Repeat solves the issue of having to resend all the packets within the window should that be an error. In fact, Selective Repeat only resends the packets where there has been an issue which in turn increases the throughput.

Question 7 – Experimentation with *iperf*. For each value of window size, run the experiments for **5 times** and write down **average throughput**.

	Average throughput (Kilobytes per second)
Window Size (KB)	**Delay = 25ms**
1	13.8
2	26.2
4	27.9
8	63.2
16	87.6
32	78.7

Question 8 - Compare the throughput obtained when using "Selective Repeat" and "Go Back N" with the corresponding results you got from the *iperf* experiment and explain the reasons behind any differences.

The results obtained from the iperf experiment show that the average throughput for both Selective Repeat and Go Back N are higher. One way to explain this, is that by default, iperf uses TCP which is overall slower than UDP (that we've implemented through sliding window protocols).

In fact, UDP allows for a continuous packet stream because of the way packets are acked which is not the case for TCP. In addition to that, TCP packets are heavier (bigger header size) and the TCP protocol employs a flow control algorithm that can slow down the transmission.

Last but not least, UDP is a connectionless protocol whereas TCP is connection oriented and therefore proceeds to a prior handshake that might also slow down the transfer and give a lower average throughput