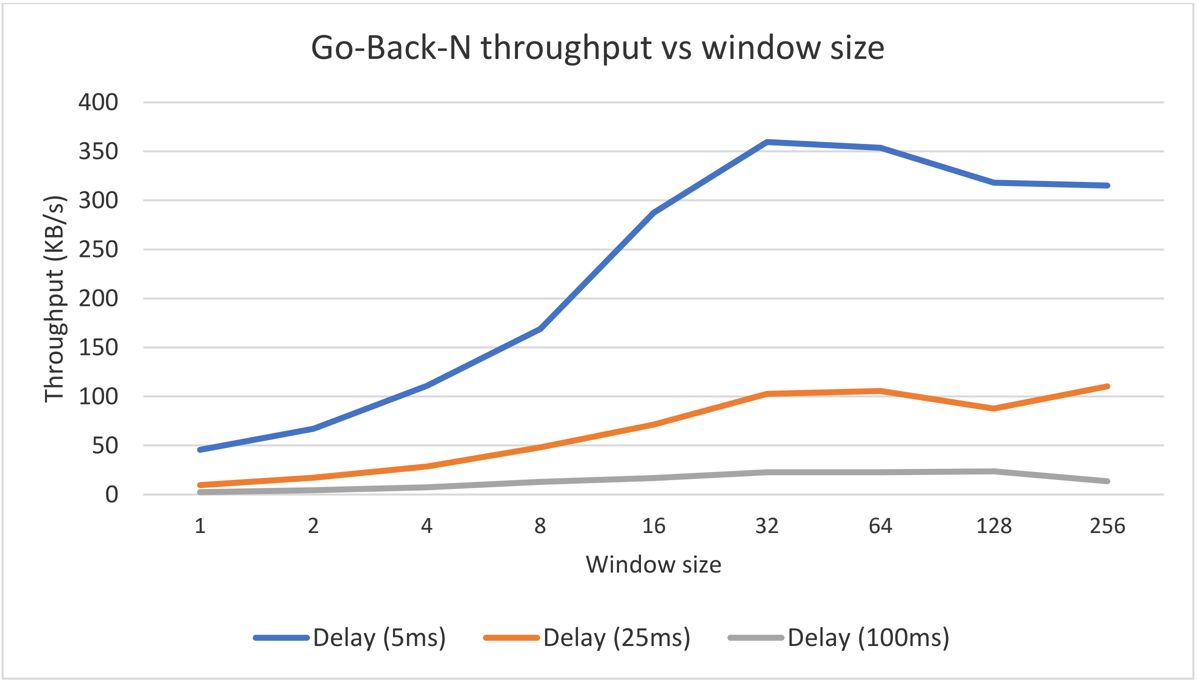
Computer Communications and Networks (COMN) 2019/20, Semester 2 Assignment Part 2 Results Sheet

Forename and Surname:	Joao Maio
Matriculation Number:	s1621503

Question 1 – 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	45.62	9.62	2.43
2	66.94	17.11	4.52
4	110.96	28.61	7.43
8	169.05	48.02	12.95
16	287.43	71.50	16.68
32	359.51	102.75	22.76
64	353.72	105.62	22.86
128	317.96	87.63	23.67
256	314.99	110.33	13.51



Question 2 – Discuss your results from Question 1.

Window sizes increase throughput when compared to the "simple" implementation from Part 1b. The benchmark for this protocol is a window size of 1, which is similar to Part 1b (save for possible threading overhead).
The increase in throughput appears to be consistent across the different delays, and is also proportional – for example, the peak throughput at 5ms delay is roughly 360KB/s with a window size of 32, and the throughput at 25ms delay is roughly lower at 100KB/s for the same window size – this change is not an exact factor of 5, but is close enough considering performance may have peaked for the 5ms delay connection. At 100ms delay, peak performance is sixteen times lower than 5ms and four times lower than 25ms – for 25ms, this is exactly the multiple difference in delay (25ms × 4 = 100ms).
There are diminishing returns, however, with the most throughput happening at a window size of 32, and subsequent window sizes experiencing either no significant change or a drop in performance, along with a much higher utilization of the network.

Question 3 – 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 Size	**Delay = 25ms**
1	9.60
2	16.73
4	27.07
8	41.57
16	61.28
32	67.97

Question 4 - 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.

"Selective Repeat" appears to be slightly behind "Go Back N" in terms of throughput at all window sizes.
It may be the case that Selective Repeat has to wait longer for a single intermediate packet to be ack'd before the base can be incremented, where "Go Back N" would simply re-transmit all packets from the base instead of "selectively" sending those missing packets, and so waiting a shorter amount of time before re-transmitting the full window.
In this case, "Selective Repeat" could be stuck waiting for two different packets in the window that will be sent separately, taking roughly twice as long to ensure that they are received, whereas "Go Back N" would simply re-transmit the full window in that same time. "Go Back N" will more likely cover both of these missing packets in the same timeframe and so be faster. "Selective Repeat" is much more efficient with its network usage, even if it suffers slightly in terms of throughput.

Question 5 – 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	13.76
2	21.80
4	46.32
8	75.48
16	81.80
32	106.28

Question 6 - 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.

In testing, *iperf* is faster than both "Selective Repeat" and "Go Back N".
From subsection 3.5.4 in Kurose & Ross, TCP's error-recovery is described as a hybrid of "Go Back N" and "Selective Repeat". This hybrid approach has the benefits of the throughput of GBN while maintaining the network efficiency of SR. Most importantly, TCP has a more advanced system for handling acknowledgements (cumulative acknowledgements) than the previous two, which permits fewer re-transmissions, resulting in less time being spent on a data transfer. Because of this, TCP can achieve faster speeds while avoiding network congestion.
In this experiment, it looks as though the normal TCP congestion controls were likely not triggered because the network is being used exclusively by *iperf*. Also, with both the sender and receiver's window sizes overridden, both sides were likely able to sustain the load, not triggering TCP flow control, and resulting in an unexpected speed increase over SR and even GBN.
Ultimately, the implementations of Part 2a and Part 2b may have lower performance than expected due to not being well-optimized – a likely suspect is the use of the RandomAccessFile Java class to write directly to the received file as soon as packets within the window are received.
This may be a happy accident that showcases how network throughput can be affected if the client has low disk-writing performance: a situation which GBN and SR are ill-equipped to deal with since they lack *flow control*, or 'back-pressure', to throttle a sender's transmission speed, overwhelming the receiver.