Computer Communications and Networks (COMN) Course, Spring 2011

Coursework: Results Sheet

Forename and Surname:	Stephen McGruer
Matriculation #:	S0840449

Question 1 – Experiments with Stop-and-Wait:

Retransmission timeout (ms)	Number of re-transmissions
10	46,532 (22.708 KB/s)
20	29,496 (23.499 KB/s)
30	10,701 (23.977 KB/s)
40	4,113 (23.741 KB/s)
50	136 (23.515 KB/s)
60	83 (23.330 KB/s)
70	63 (23.300 KB/s)
80	74 (23.035 KB/s)
90	135 (22.605 KB/s)
100	1841 (21.179 KB/s)

Question 2 – How does the throughput behave when the retransmission timeout increases and why?

As the re-transmission time-out increases from 10ms, the throughput also increases until it reaches a peak (at a time-out of 30ms), then decreases after this. I theorise that this is caused by the changing ratio of re-transmission time-out to number of packets sent.

A low re-transmission time-out causes many *false-negatives* to be sent – packets that are re-transmitted not because their ack was lost or even delayed, but only because the time-out occurred before an ack packet could be received. This large number of *false-negatives* slows down the throughput, due to the time costs of sending, receiving and processing the duplicate packets.

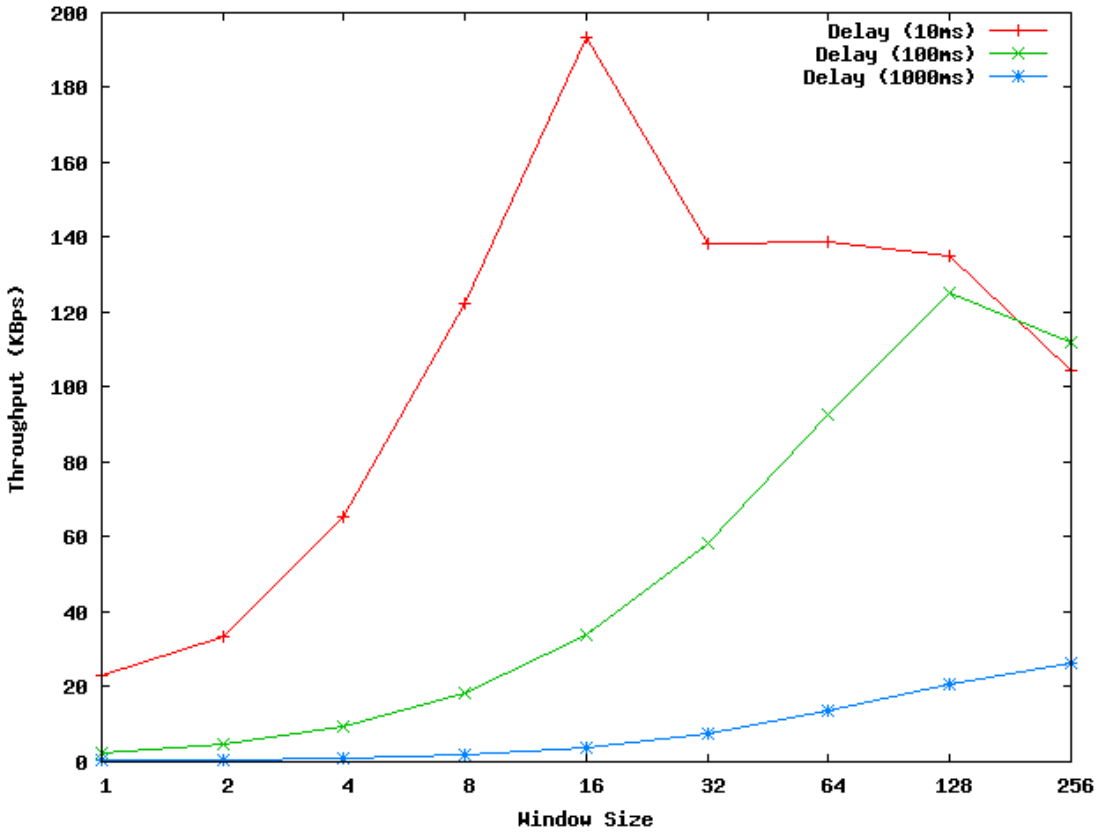
The opposite case is when the re-transmission time-out gets large, which causes an overly long delay when a packet or its ack is actually lost – the sender does not receive an ack packet and waits a long time before sending another packet to the receiver.

Therefore, since there are conditions that cause delay with both a large and small re-transmission time-out, there exists a ‘sweet-spot’ for the time-out where the ratio of re-transmitted packets to the delay on a truly lost packet is optimal. In the case of my experimentation this ‘sweet-spot’ was at 30ms, and so this is the value I will use for Question 3.

Question 3 – Experiments with Go-Back-N:

	Throughput (Kilobytes per second)		
Window Size	**Delay = 10ms**	**Delay = 100ms**	**Delay = 1000ms**
1	23.13	2.46	0.25
2	33.33	4.78	0.5
4	65.64	9.44	0.99
8	122.20	18.42	1.70
16	193.64	33.69	3.69
32	138.12	58.47	7.43
64	138.93	92.8	13.57
128	135.25	125.41	20.88
256	104.39	111.9	26.54

Use the results in the above table to make the following graph:



Question 4 – Explain your results from Question 3.

These results show that the performance of Go-Back-N transmission relies on a careful balance of two factors – the window size and the propagation delay.

In general, a larger window size will increase the transmission throughput, by allowing more packets to be sent in each grouping. A too large window size, however, can actually cause the throughput to drop as the receiver becomes overwhelmed with incoming packets, being unable to ack packets quickly enough for the sender to move the window forwards at a reasonable rate. Whether or not a window size is 'too large' depends on the propagation delay in the channel. The larger the propagation delay is, the higher the window size at which the throughput peaks, as the receiver is less overwhelmed by a large number of packets due to their slower incoming speed.

The data collected from this experiment backs this theory. With a delay of 10ms there is a 'peak' at a window size of 16. This rises to 128 when the delay is 100ms, and with a 1000ms delay I did not actually reach any peak.

Question 5 – Experiments with Selective Repeat

Window Size	Throughput (Kilobytes per second) Delay = 100ms
8	19.26
16	36.94
32	74.06
64	120.42
128	144.53
256	138.87

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.

The results I obtained from using Selective Repeat were slightly faster than Go Back N. This makes sense, as the main problem with Go Back N that Selective Repeat fixes is the re-sending of out-of-order packets that have already been sent and received successfully by the Receiver. That is, if packets 1, 2, 3 and 4 are sent to the Receiver in Selective Repeat and only 3 and 4 are received, then the Sender will only have to re-send packets 1 and 2 (assuming the Receiver acked packets 3 and 4). In Go Back N the Sender would have had to resend all four packets.

Question 7 – Experiments with iperf

Window Size (KB)	Throughput (Kilobytes per second) Delay = 100ms
8	17.4
16	26.3
32	40.4
64	44.1
128	59.0
256	44.2

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 UDP-based Go Back N and Selective Repeat experiments were far quicker than the results I obtained from experimenting with the TCP-based *iperf*. This is not unexpected, as TCP is a far slower transfer method than UDP, sacrificing speed for reliability. TCP includes checks for data corruption, and also attempts to avoid flooding the data channel with packets, unlike in our UDP implementations (where we simply push as many packets down the connection as possible.)

Notes:

The commands I ran for this experiment were:

iperf -s -f K -M 1024 -w {N}

iperf -c 127.0.0.1 -f K -M 1024 -w {N} -n 2.7M -F testfiles/cwk_testfile.jpg

Where *{N}* was the window size in bytes, i.e. 8192 etc.

It is important to note that I came across a small problem when using iperf that means that my experimental data for the window size of 128KB and 256KB may not be entirely correct. When using a window size of 128KB, I ran into a warning that the window size had been set to 244KB – **less** than double 128. I then tried using a window size of 256KB, which gave a warning that the window size was again set to 244KB. I believe this means that there is some sort of ‘cap’ on the window size, and as such my results for a window size of 128KB cannot be entirely trusted and my results for a window size of 256KB are likely to be completely wrong.