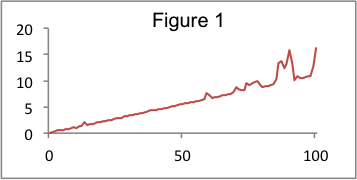
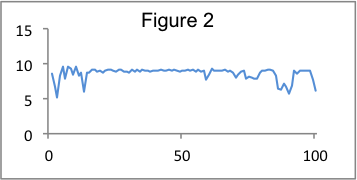
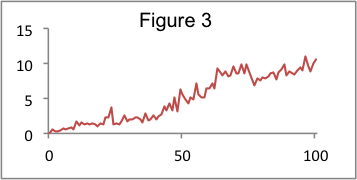
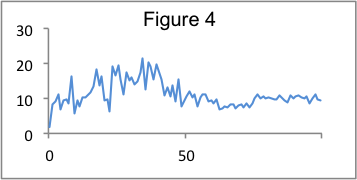
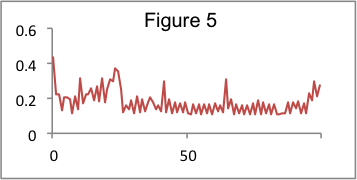
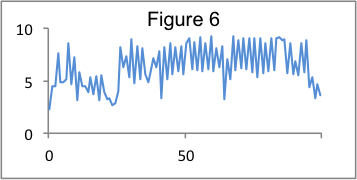
**SENG 513 Deliverable 2 - Performance Testing Report**

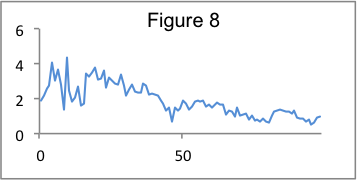
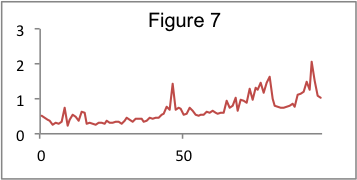
Performance testing was performed for two operations: uploading, and accessing a user’s feed. Our tests are provided in the *test/performance\_testing/* directory. In order to perform these tests we made use of bulk operations to clear and set up our database as required. We also used the instructor-provided JSON objects as well as generating our own JSON object as can be seen in *makePhotoJSON.js*. In addition, in order to test the effect that the number of images in a feed has on the performance of that feed, we created *“Feed(0Followers)\_NumImagesVsResponseTimePart1.js”* which uses our generated photos JSON object to set up the database so that 99 users will have a varying number of photos from 1 through 99. This setup had to be done separately as our bulk photo request would time out otherwise.

**Results**[[1]](#footnote-1)

Response time increases linearly with an increase in the number of concurrent requests on a feed that has no images (Figure 1). Throughput, measured in requests per second, remains relatively constant (Figure 2). This behaviour is expected and indicates that while our server is not overwhelmed it will provide a relatively consistent level of performance for each user request.[[2]](#footnote-2)

The effect that increasing concurrent requests has on response time and throughput for uploading is shown in Figure 3 and 4. Figure 3 is similar to Figure 1 in that response time increases linearly with an increase in the number of concurrent requests. According to Figure 4 there is an initial increase in throughput, however, it decreases and stabilizes at the tail end of the x-axis. Again, this indicates that our server handles these concurrent requests fairly and as efficiently as possible.[[3]](#footnote-3)

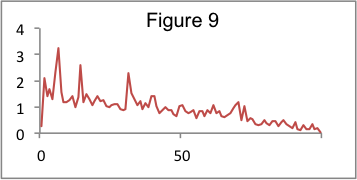
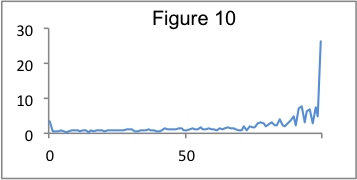
In Figures 5 and 6, we show how the number of people a user follows impacts the response time of that user’s feed. Both the response time and throughput remain relatively constant with small variances that can be attributed to network irregularities. This is a reasonable result given that the number of follows a user has should not impact how they generate their feed. This is due to the fact that we store the current feed for each user in the database, rather than generating it dynamically on the page load based on who the user is following.[[4]](#footnote-4)

Figures 7 and 8 display how the number of images a user has in their feed impacts the feed’s response time and throughput. We can see a relatively linear increase in the response time, as well as a corresponding decrease in the throughput as the number of images is increased. This is due to the fact that every image in the feed requires an extra database query to gather info about that image. These effects would be compounded by having to actually load the images in, which we will demonstrate next.[[5]](#footnote-5)

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Concurrent Requests** | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| **Response Time (ms), No Download** | 128 | 224 | 396 | 414 | 447 | 627 | 666 | 793 | 778 | 957 |
| **Response Time (ms), Download** | 1496 | 3892 | 6598 | 8563 | 12415 | 16670 | 25846 | 38959 | 44534 | 45263 |

Table 1. Effect of Concurrent Requests on Response Time of Feed with No Downloaded Photos and of Feed with Downloaded Photos (Each Feed Contains 5 Photos)

Table 1 compares the response time of the feed to increased concurrent requests when all of its component photos are downloaded, as would be seen in a web browser, and when all of the photos are not downloaded, as would occur in curl or node.js http requests. These results show that downloading the photos represents the majority of the total response time, serving as a bottleneck in accessing a user’s feed, despite the fact that it would be hidden by naive performance testing which does not download the images on each page.[[6]](#footnote-6)

Lastly, Figures 9 and 10 display the performance of photo uploading as the number of users following the uploading user increases. These results initially surprised us, as they showed that the number of followers had a negligible impact on the upload time. This is surprising as photo uploads must update the feeds of all of that user’s followers, a task which should take a relatively large amount of time. However, digging a little deeper, we came to the realization that the request returns immediately after the photo finishes uploading, but prior to the updating of all user feeds. That feed updating activity occurs asynchronously on the server, after the response has already been sent, making our upload process very efficient.[[7]](#footnote-7)

**Discussion**

Initially, it was expected that both the number of concurrent connections and a greater number of followers would lead to slower response times. Another variable thought to have a negative impact on performance was the number of photos to be uploaded.

As expected, there appears to be a positive linear relationship between concurrent requests and response time. Similarly, as more photos are included on a feed page, it will load much slower. However, the number of followers does not have a significant impact on the performance. As described in the results, this is because the database queries involved in the upload operations are performed asynchronously and the feeds are stored in the database as a list, instead of being directly tied to the number of followers.

Our performance tests served a useful purpose because they provided a clear indication of the factors that significantly impacted performance, such as the number of concurrent requests and downloading photos when accessing a user’s feed. The tests also revealed that factors that were expected to be significant, such as the number of followers, actually had a negligible effect on response time, demonstrating the importance of performance testing to both verify and challenge our beliefs about the system.

1. In every result the first figure’s y-axis is for response time (in seconds), and the second is for throughput (in requests per second). The x-axis is the controlled variable such as number of followers, number of images, or number of concurrent requests [↑](#footnote-ref-1)
2. Feed(0Followers0Images)\_ConcurrentRequestsVsResponseTime.js [↑](#footnote-ref-2)
3. Upload\_ConcurrentRequestsVsResponseTime.js [↑](#footnote-ref-3)
4. Feed(0Images)\_NumFollowersVsResponseTime.js [↑](#footnote-ref-4)
5. Feed(0Followers)\_NumImagesVsResponseTimePart2.js [↑](#footnote-ref-5)
6. Feed(0Followers5Images)\_ConcurrentRequestsVsResponseTimeWithImageDownload.js and Feed(0Followers5Images)\_ConcurrentRequestsVsResponseTimeWithoutDownload.js [↑](#footnote-ref-6)
7. Upload\_NumFollowersVsResponseTime.js [↑](#footnote-ref-7)