Most of my testing involved unit testing with each portion of the code. The way I would test my code by essentially overloading my cache with files. I did this by curling 5 different PUT request files into the cache and checking if the cache was being updated in both size and contents as well as observing if the file got written to disk. Also I would PUT the same multiple times into the cache to observe if the file would be updated and checking if the cache size would stay the same. I would also check if my code properly disabled caching by curling as normal checking if the files were correctly written to disk when no caching flag was given. I would check GET requests by curling PUTs and requesting said PUTs and seeing if anything was written to disk or not. I know I was getting the write result when nothing was being to disk yet I was still getting the information I was requested from the GET. I would also manually put files onto my server side and check if my GETs were properly updating the cache when if the file on the disk was not on the cache. Lastly I would check the logfile if the -l option was given to see if the correct message was written if the file was in the cache or not. GETs would write “was in cache” if the requested info was in the cache and vice versa if not. PUTs would write “was in cache” if the file was being updated, and it would write vice versa if the file was not. All of this was done by constantly curling the correct order of PUTs and GETs to check for each corner case.

I did a latency experiment with my servers to check to see how much faster the second of two consecutive GET commands would be. It turns out that when I GET from a server without caching the speed is actually faster than when I GET from a server with caching. This is probably because of how I decided to store my file contents into the queue. Since I used a deque and didn’t allocate memory, the caching server probably took a little more time than the non caching server due to the deque having to adjust the size for me. However, I did notice speed up

from the second GET to the caching server by a very significant margin of almost twice the speed. After the first GET request to the caching server, the server had the file in the cache so when the client GET requested again the time was much shorter since the file was already in the cache. This makes sense as instead of having to read and then write from a file, we can simply just look up the file in the cache, which should be the most recent update to the file, and just write that immediately. Also from how my code is setup, when you write from a cache, you write the entire file all in one go instead of writing incremental buffers to the file when reading from the disk.