This report analyses the performance of three implementations of the same program - threaded, multiprocessing and serial. The task of the program is to search for a keyword on some server (here Dominiqu's) that will give back a list of urls of files (20 or less) that contain the keyword. Then the program queries each of the url, gets the whole file, removes stopwords, calculates the frequency of each word, ranks the words and then assigns a score to the document as the inverse of the rank. (So one query is defined as accessing one url and calculating it's score.) Then, in a decreasing order of score, it outputs the url, its score and a context around the keyword.

The time of the execution is measured with Python's time class. It starts measuring after the list of urls have been retrieved and before each url is processed. It stops measuring the time when all urls have been processed and before arranging them in the order of the score. Thus the time measured is of the part of the program that is executed in parallel in threaded and multiprocessing implementations. The best time and average time of 3 runs is calculated. The number of queries per second is calculated as number of queries divided by time it took to process them.

The experiment was done on a Intel(R) Core(TM)2 Duo CPU P8600 with speed 2.40GHz each. The time of the day was around 8pm. Internet connection was Smith College residential Wifi for the first two experiments.

In the Figure 1 we can see the results of number of queries per second, whith the keyword searched being "love". The multiprocessing implementation of the program performs the best with 9.4 queries per second as the best result among three runs and 7.9 queries per second on average. Not so much surprisingly anymore, the serial version performs better than the threaded version (but worse than the multiprocessing one). This probably is due to the Python Global Interpreter Lock, that allows only one thread to perform at a time.

These results are specific for the keyword "love". In Figure 2 are results for different keywords. When keywords "love", "car" and "joy" were searched, 20 urls were returned, for the keyword "blablabla" none were returned, and for the keyword "christmass" 4 urls were returned.

Again serial version performs better than the threaded version every single time. The multiprocessing implementation outperforms the other two, except for the keyword "joy". The explanation for this anomaly could be that it happened that at the time the experiment

was run and specifically the search for the keyword "joy" was performed, there was a some kind of lag in the Internet connection.

Also in Figure 2 it seems that performance is better if less urls are processed (as in the case with keyword "christmass"). However there is too few data points to be completely sure. To explore it in more detail it would require to be able ask the server to return arbitrary number of results (not 20 or less as it is now).

I also wanted to see how performance changes with different Internet connection speeds. The experiment was run on a wifi connection in the Engineering building (Green Box) (the speed there was the slowest), wifi connection at residential house (medium speed) and ethernet connection at the same residential house (the fastest). In Figure 3 differences and dependence of the performance on the Internet speed is clearly visible. The faster the Internet connection the better performance. The multiprocessing implementation outperforms the other implementations at medium and fast Internet speeds, however the results at low Internet speed are very close. What is different from the other graphs is that at the highest Internet speed the serial version is slower than the threaded version.

In summary, the multiprocessing implementation almost all the time outperforms the threaded version, which is expected due to the Python Global Interpreter Lock, which allows to execute only one thread at a time. It is relevant to this program, since part of the program (maybe even the largest part) is spent in a "cpu bound" mode (that is when calculating the score of the document). Even the serial implementation outperforms the threaded version, which is consistent with my results in homework 2.

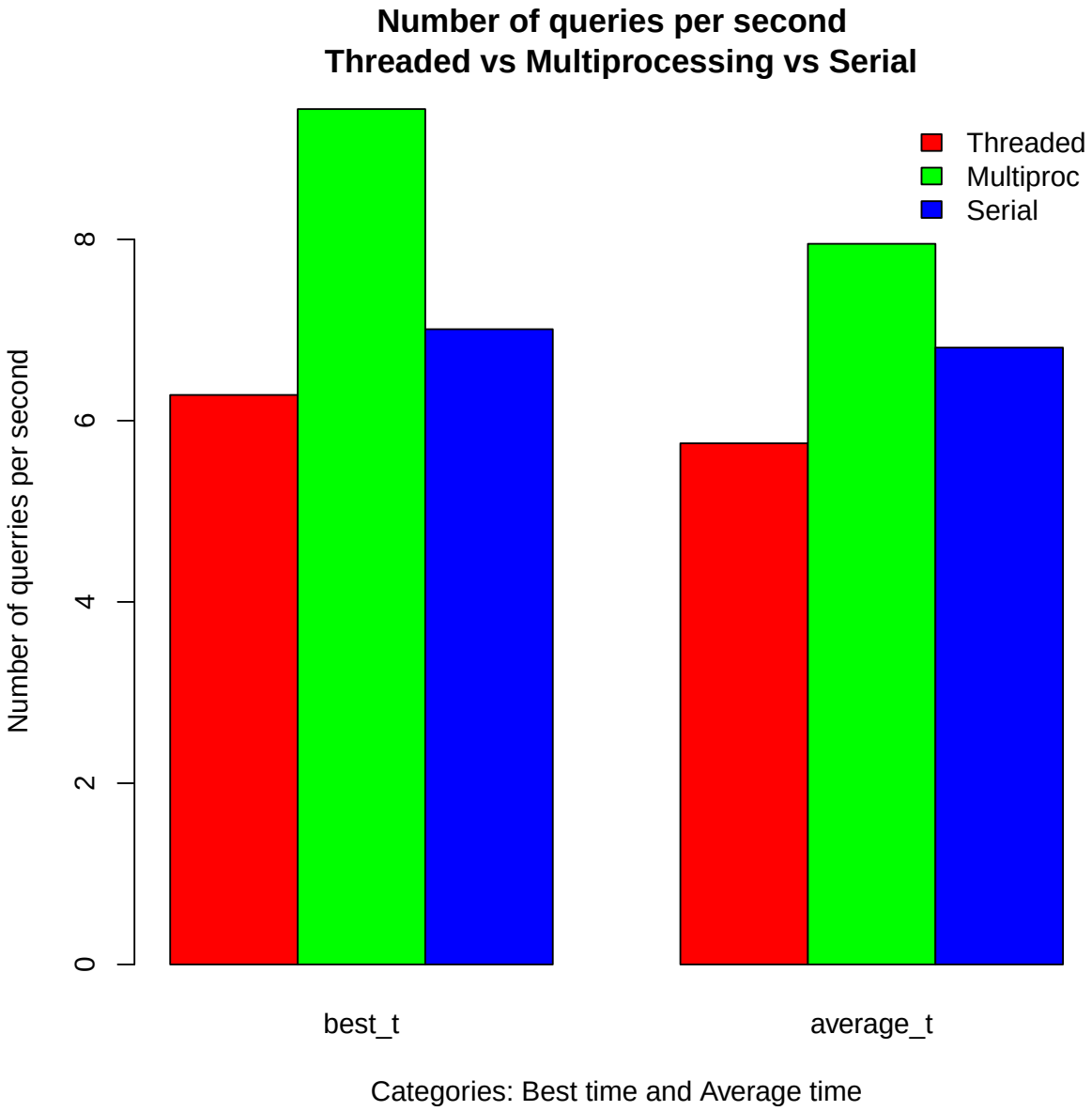

Figure 1: Number of queries per second. The best and average time for retrieving 20 urls with keyword "love".

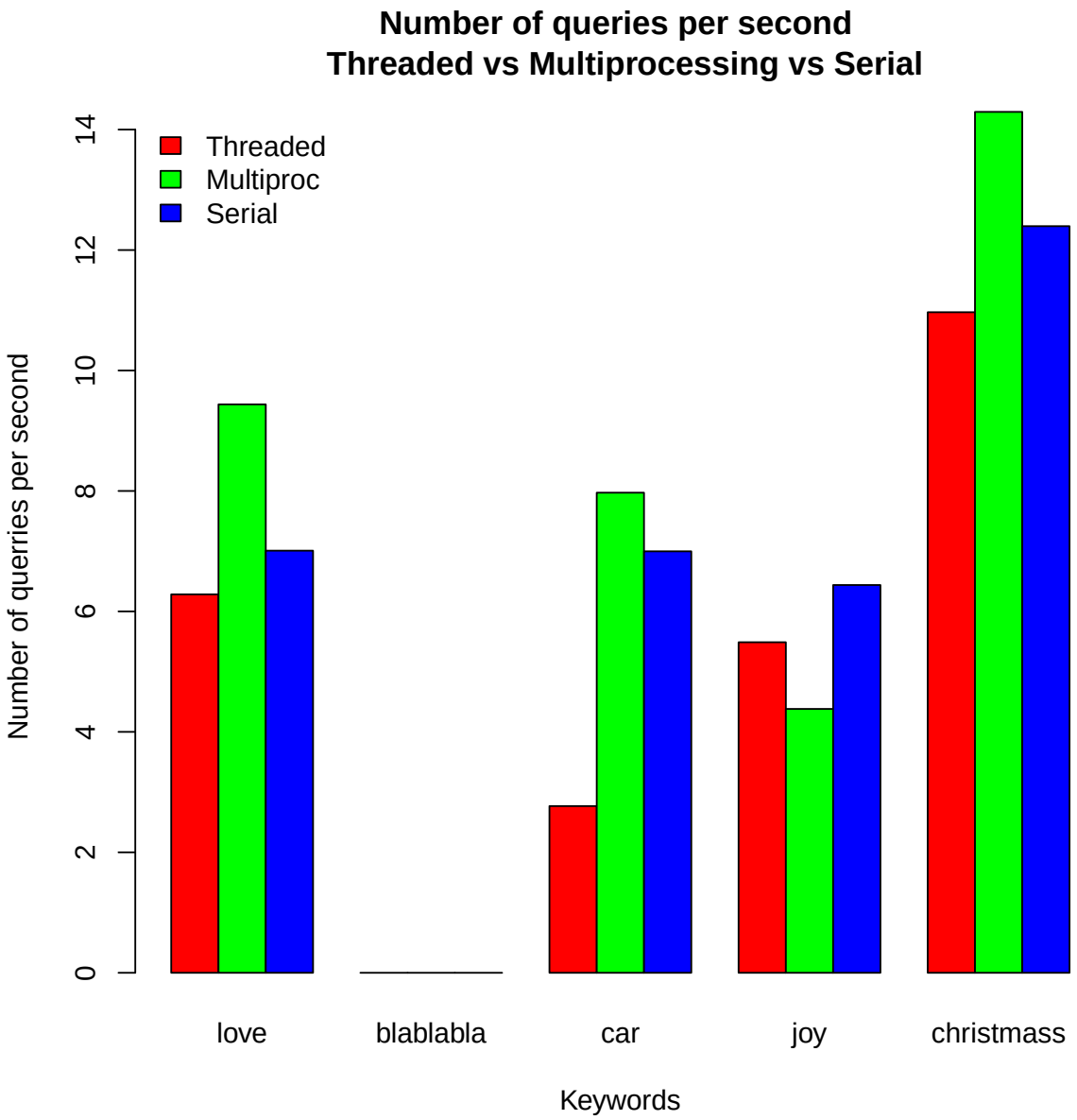

Figure 2: Number of queries per second. The best time for different keywords.

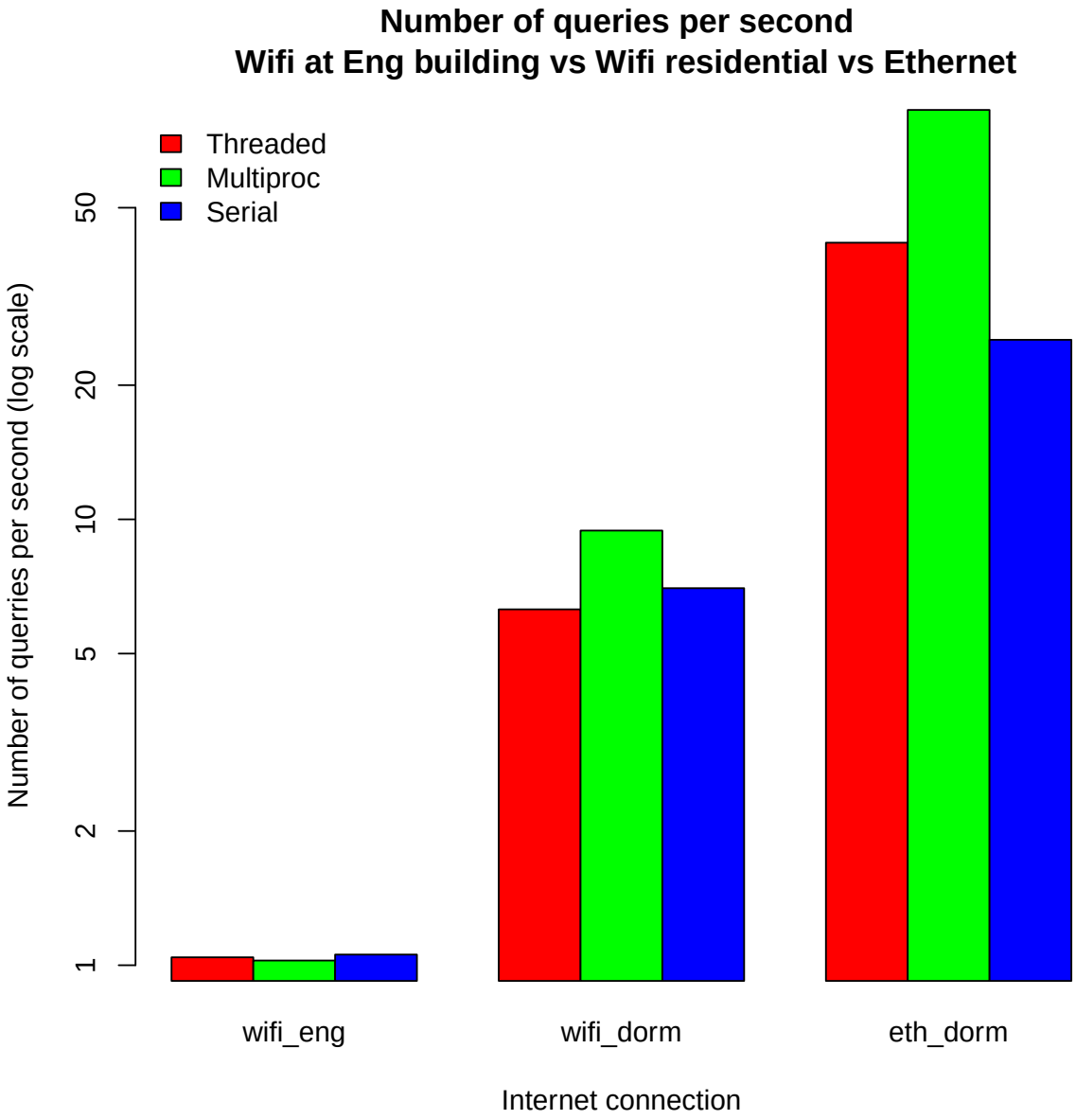

Figure 3: Number of queries per second. The best time for the keyword "love", with experiment performed with different Internet connections.