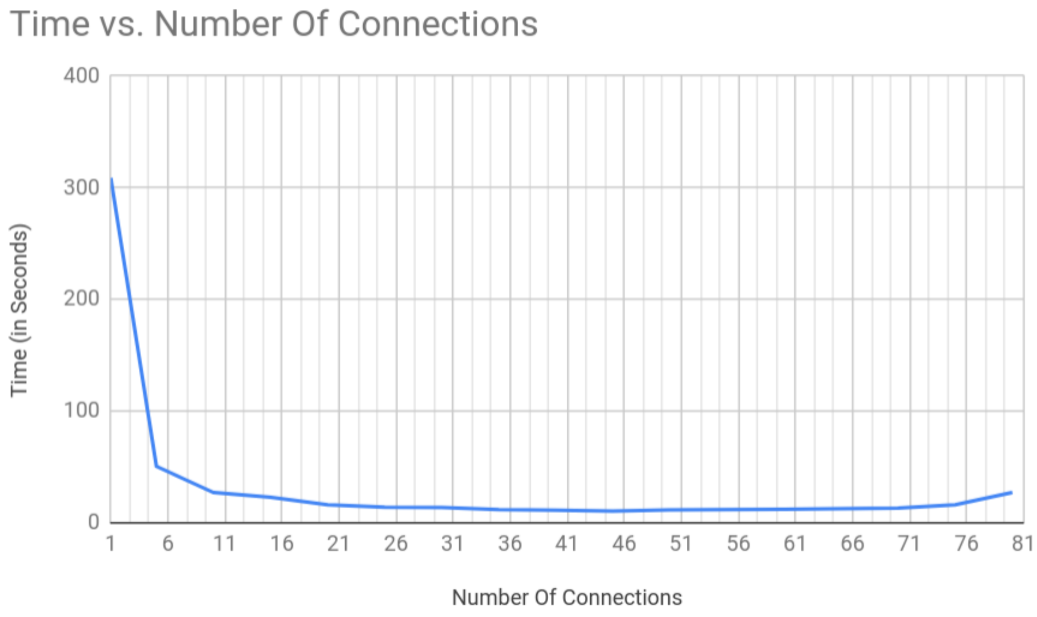
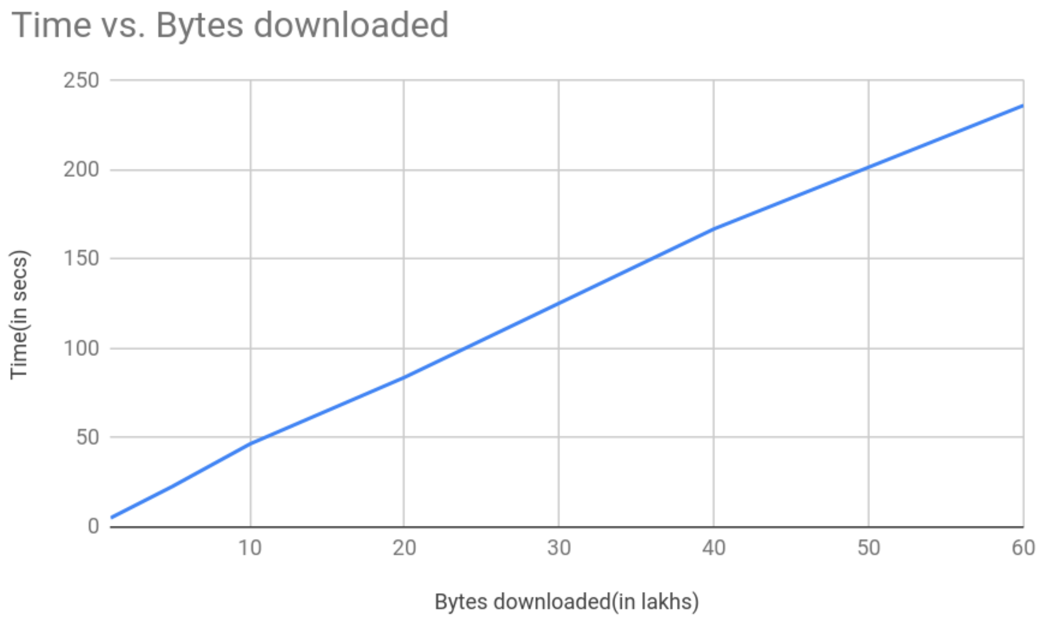
- 1.
  This is implemented in 1.py. It continuously receives data until a blank string is received.
- 2.
  This is implemented in 2.py. I downloaded the file in chunks of 10KB. I found that the last packet may not be of the same size of 10KB. Thus I needed to know what is the file size and then I could reduce the chunk length for the last message. So before downloading I send a HEAD request to get the content length of the file.
  Also sometimes I was seeing that the socket somehow closes in between. After this, any received data is just a blank screen. I also handled this case by restarting the connection
- 3.
  - For introducing multiple connections, I used threads. I made multiple threads for multiple connections. To keep track of progress I made a synchronized object containing the current starting byte that should be downloaded. After a thread has accessed it, it is incremented.
  - I also maintained a synchronized array where each thread would keep appending the messages they received with the starting byte. After all bytes are downloaded the array is sorted and answer is written to stdout.
  - The download time first decreases and then increases as number of connections are increased. The decrease is due to the fact that more chunks are getting downloaded in parallel. The increase afterwards is due to the overhead of threading. The system is not able to handle so many threads and since there is synchronization in some elements, the overhead for more threads also become more. Below is the trend I saw with vayu server. Data points are in statsNumConnections.csv file



  - Bytes downloaded vs time taken trend seen on vayu server with 1 connection. Data points are in bytes.csv file



  - The time taken by Norvig is more than Vayu, indicating Vayu is relatively faster, maybe since its in Delhi whereas the Norvig server may most probably be in USA, something I also saw in an IP lookup service. In most cases, I found out that mixing both would only increase the time of download. But sometimes I was getting a sped up, indicating congestion also play a role in the download time. In a mix, most packets will be downloaded by Vayu (since its faster), but since we are sharing it among different servers, there is less congestion on one particular server from our side and this might also lead to faster times. The bottleneck mainly lies in the RTT latency of packets and also in the congestion.
  - Since I dynamically allocate chunks to connections, faster connection would end up downloading more chunks. So the greater proportion of the file comes from faster connection.

- 4.
  I set the timeout to 25 seconds. To make the code fault tolerable, I put exception handling whenever I try to connect the socket or receive data. If I get an error, I download the chunk again by restarting the socket.
  Usage: python 4.py $(csvfile) > (outputfile)$
  In my experiments, I used objects.csv as input.
  Time taken is printed to stderr file, which is terminal output by default.
  I also print some statements, when a timeout happens or a connection is being established, to stderr. A sample dump is in $stderrFile\_10\_4.txt$, which had 10 connections to vayu and I shutdown the net 4 times.
  For checking if the file downloaded is right, I ran a diff on the downloaded file with the original one, along with MD5. MD5 is computed and compared in md5.py.