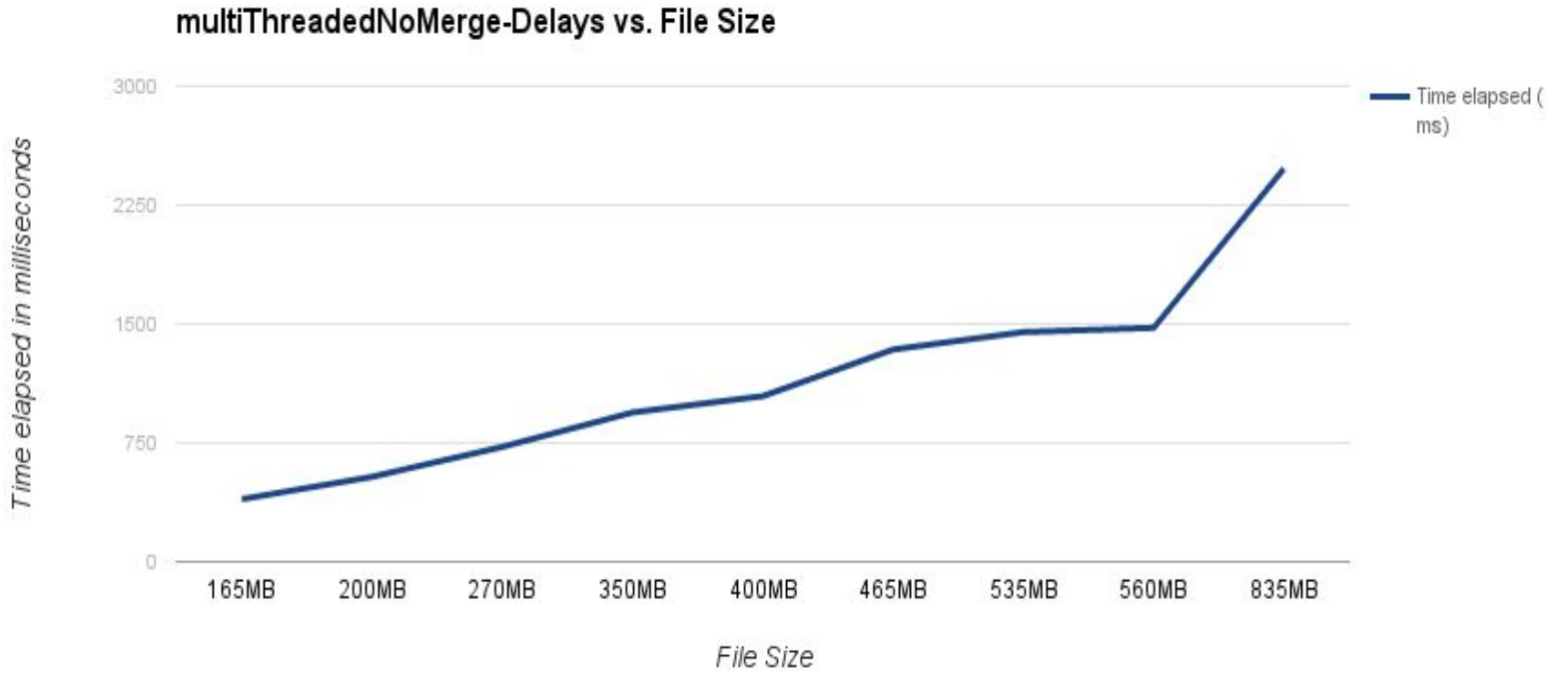
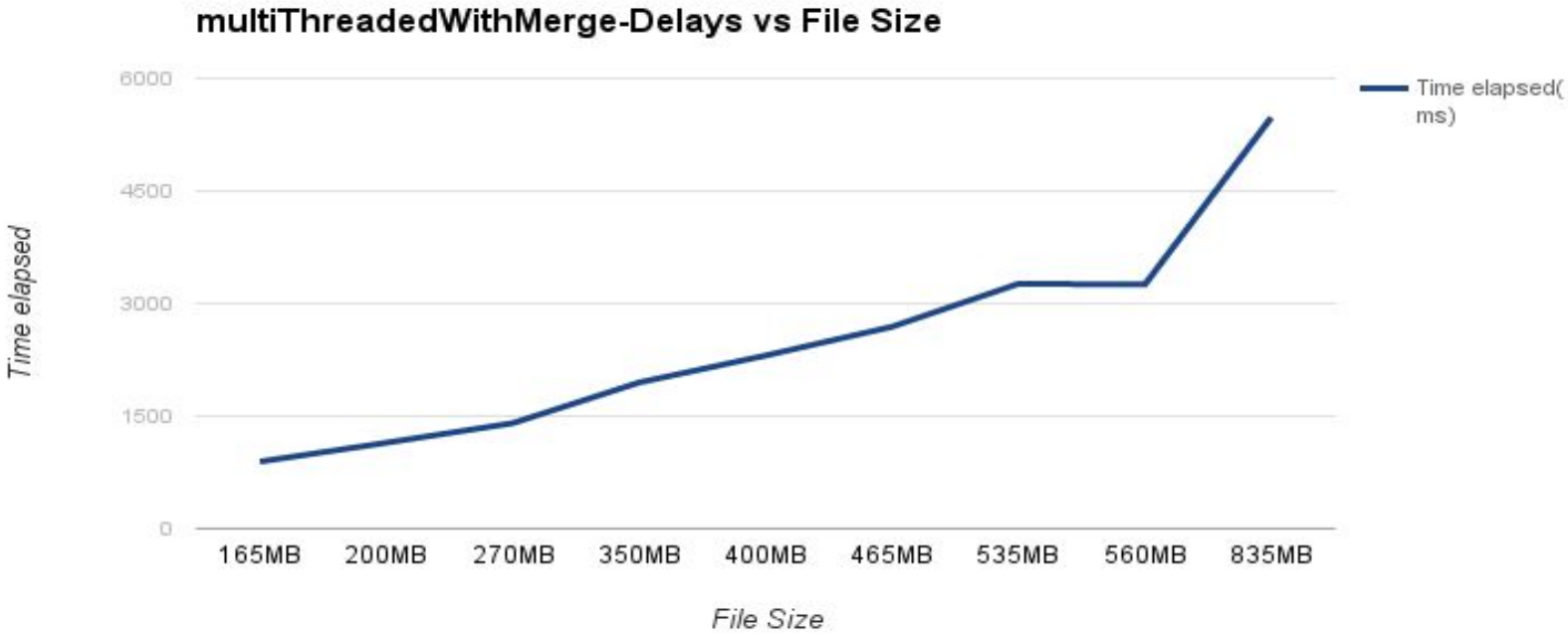
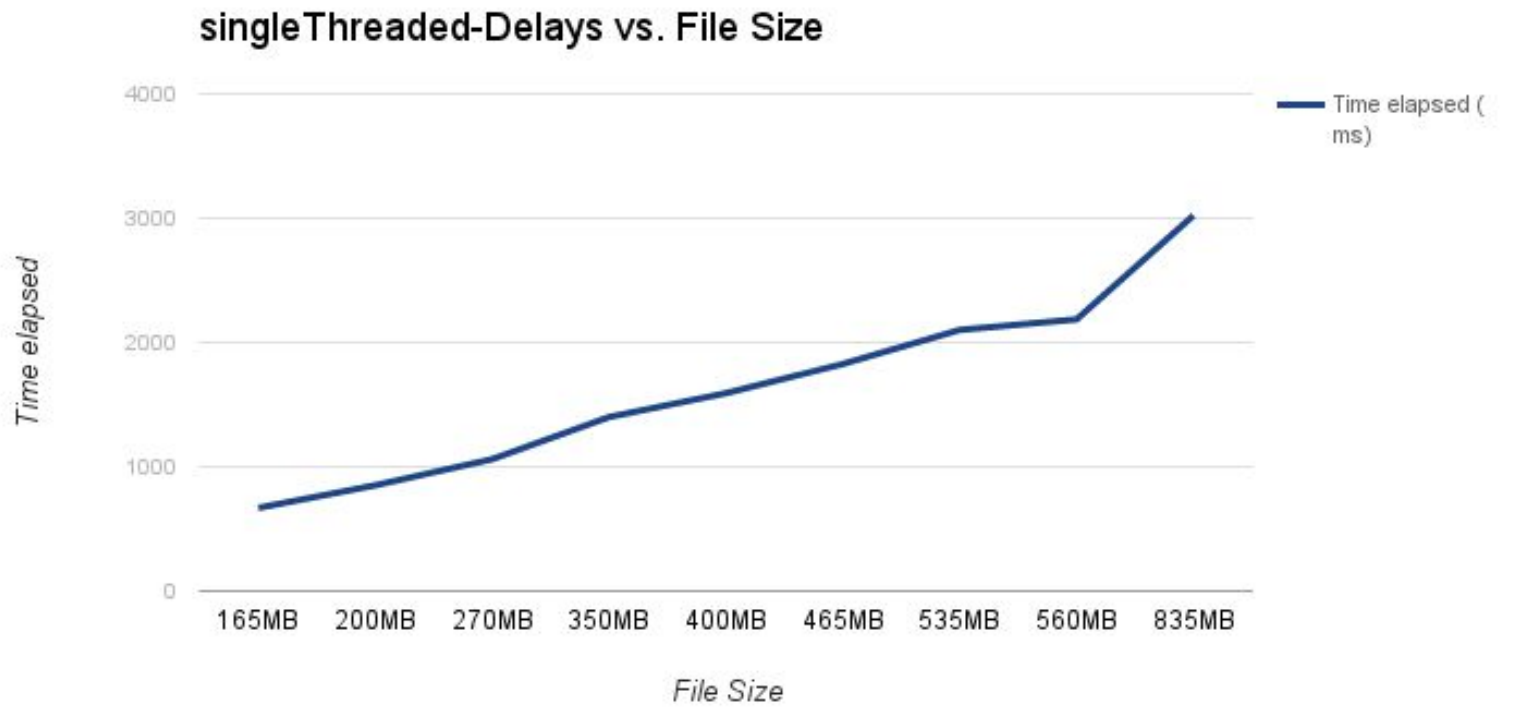
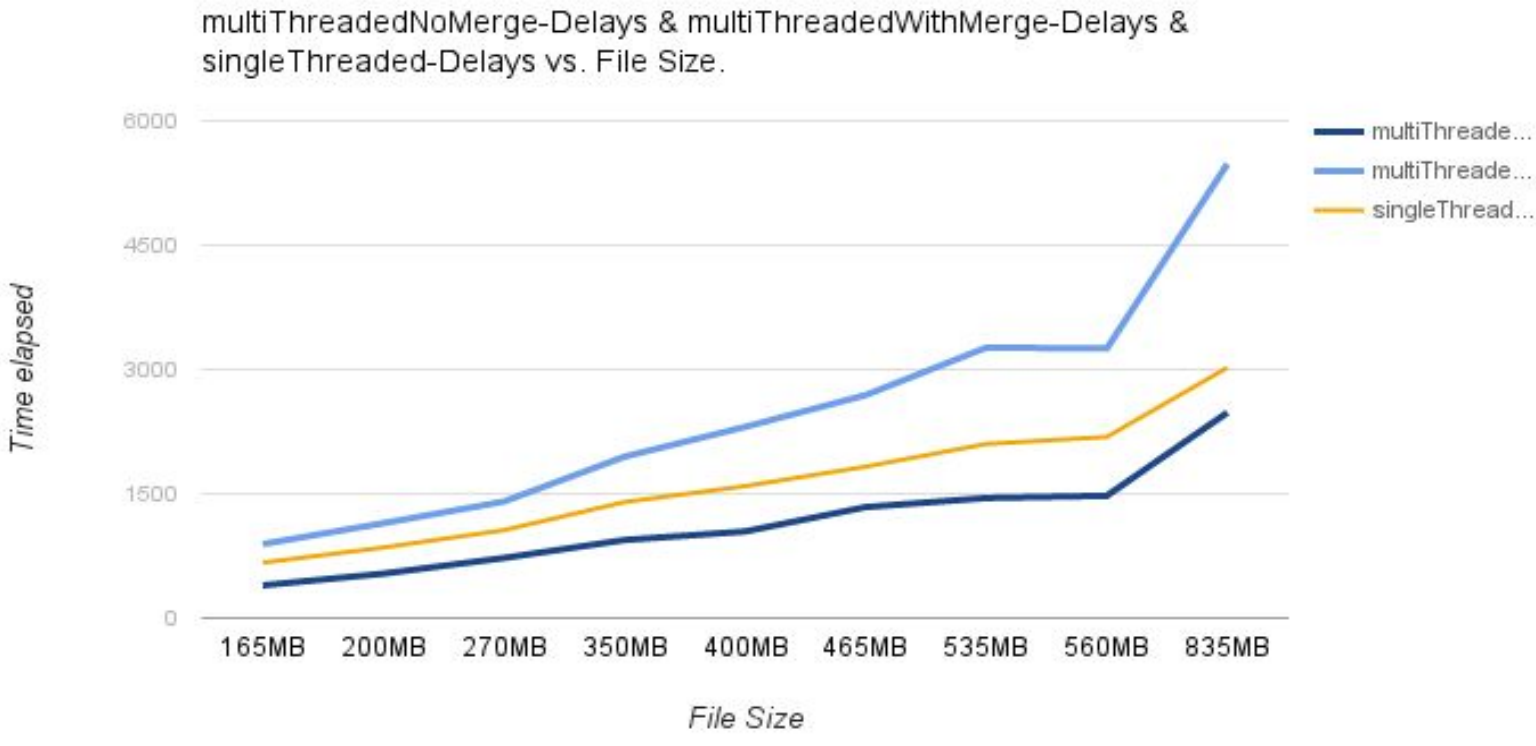

multiThreadedNoMerge-Delays vs. File Size
Time elapsed in milliseconds
3000
2250
1500
750
0
165MB
200MB
270MB
350MB
400MB
465MB
535MB
560MB
835MB
File Size
Time elapsed (ms)


multiThreadedWithMerge-Delays vs File Size
Time elapsed
6000
4500
3000
1500
0
165MB
200MB
270MB
350MB
400MB
465MB
535MB
560MB
835MB
File Size
Time elapsed(ms)


singleThreaded-Delays vs. File Size
4000
3000
2000
1000
0
Time elapsed
Time elapsed (ms)
165MB
200MB
270MB
350MB
400MB
465MB
535MB
560MB
835MB
File Size


multiThreadedNoMerge-Delays & multiThreadedWithMerge-Delays & singleThreaded-Delays vs. File Size.
6000
4500
3000
1500
0
Time elapsed
multiThreade...
multiThreade...
singleThread...
165MB
200MB
270MB
350MB
400MB
465MB
535MB
560MB
835MB
File Size

Disclaimer:
The following program was done on a same computer both being client and server.

Analysis of the graph

The four graphs above are represents:
a. multiThreadedNoMerge-Delays vs. File Size
b. multiThreadedWithMerge-Delays vs. File Size
c. singleThreaded-Delays vs. File Size
d. multiThreadedNoMerge-Delays & multiThreadedWithMerge-Delays & singleThreaded-Delays vs. File Size.

The x-axis displays continuous, rather than discrete, data. Each data point has an x and y value and is connected by a line. This basic line chart shows file downloading time over file size, with file size along the x-axis, and the file downloading time over the y-axis. The file downloading time seems to increase with the increase in file size. Hence, the relation between file downloading time and file size is directionally proportional.

Let,
a. multiThreadedNoMerge-Delays procedure be called as Process1
b. multiThreadedWithMerge-Delays procedure be called as Process2
c. singleThreaded-Delays procedure be called as Process3.

After analyzing all the graphs, I concluded that the Process1 was quickest among other processes. Process 2 was the slowest, whereas Process3 was in the middle.

Lessons learned in the project

I learned that files download faster when it is splitted and sent individually to the client. But the time consumed to merge the individual sent files will delay the downloading process. If files are downloaded without splitting as in the Process 3, downloading process completes at a time faster than time calculated files after merging, slower than the time calculated when files downloaded before merging.

The core lesson to be learned was if this program was actually a software as download accelerator, the files would download faster than the regular time by splitting the file into a number of sections and downloading the sections more or less simultaneously. This approach would be a huge help with large files.

Problems and adopted solutions

I did not specifically face any problem regarding this program but overall it was worth it. Coming to this point, I can clearly say how download accelerator work. While researching more about download accelerators, while I found out that Torrents work on the same way as well, breaking a file into many pieces so that it can be downloaded separately by requesting the server to allow multiple connections. Using bytestream, the files are downloaded with smaller divided sizes. The parts downloaded from each of the requests are joined together in order to make the complete original file, after the downloading of every stream completes. It uses the logic of breaking one huge task into many smaller parts.

Any other relevant inputs/comments

However, I understood that the download accelerators does not increase the download speech more than the connection can. It just optimises the way our connection is used, that is ensuring the connection to be fully used by the different downloading threads that it has created. It is just providing a method to manage downloads more efficiently and make the maximum use of the connection.