# COL 334
# Assignment 2

Kushagra Rode

September 2022

## Design

Programming Language used - Python

My design on the server side involves N threads and 2N ports. On the client there are initially N threads which are then further split in to 2 threads for each client which are used to handle the sending and receiving functions for the client.

There are 2N ports on the server side. On 1 port there are 2 sockets one TCP and one UDP. One socket on this port would function as UDP sending and TCP receiving and similarly for the other port. This is done to ensure that sending UDP and receiving UDP messages do not interfere with each other. Similarly for the TCP sockets.

On the client side also there are 2N ports in total which each client having 2 ports, again one port has both UDP and TCP socket. These are done to separate the request sending(queryServ) and broadcast receiving(respServ) functions.

Further, each chunk in my program is of size 1030 bytes this includes a 4 byte chunk number and 2 byte delimiter("$$"). Chunk number is used to uniquely identify each packet number

For handling UDP packet loss, I have made use of acknowledgements and timeout functionality. This implies that a UDP message is sent until an acknowledgement for the same is not received.

Design for Part 2 is also similar to that of Part 1

My design for both the parts can handle the case when say the client side is started first. Traditionally, we know a server starts first before the clients but if the client file is run first, my design would make it wait until a server has been started.

## Analysis

1. (a) The average RTT for all chunks across all clients for number of clients as 5 in Part 1 is 0.009530648075 seconds.

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  JUPYTER

91975@DESKTOP-2SRPFC4 MINGW64 /d/study material/sem5/COL334/ass2
$ ./2020CS10354.sh
sending initial data.....
sending initial data.....
sending initial data.....
sending initial data.....
sending initial data.....
md5 sum for 4: 9f9d1c257fe1733f6095a8336372616e
md5 sum for 3: 9f9d1c257fe1733f6095a8336372616e
md5 sum for 0: 9f9d1c257fe1733f6095a8336372616e
md5 sum for 1: 9f9d1c257fe1733f6095a8336372616e
md5 sum for 2: 9f9d1c257fe1733f6095a8336372616e
Total time taken = 1.9979195594787598 seconds
average RTT : 0.009530648075301072
```

(b) The average RTT for all chunks across all clients for number of clients as 5 in Part 2 is 0.001433880164 seconds.

```
PS D:\study material\sem5\COL334\ass2> python clients_part2.py
md5 sum for 0: 9f9d1c257fe1733f6095a8336372616e
md5 sum for 4: 9f9d1c257fe1733f6095a8336372616e
md5 sum for 3: 9f9d1c257fe1733f6095a8336372616e
md5 sum for 2: 9f9d1c257fe1733f6095a8336372616e
md5 sum for 1: 9f9d1c257fe1733f6095a8336372616e
Total time taken = 1.7005419731140137 seconds
0.0014338801647054739
PS D:\study material\sem5\COL334\ass2> python clients_part2.py
```

The RTT of Part 1 is higher. This is because of the fact that TCP was used for data transfer in part 1 which is a reliable protocol and hence has to ensure correctness of the data received, along with this TCP needs to perform a 3 way handshake which takes some time. In Part 2, UDP was used for data transfer and it being an unreliable protocol tends to be faster than TCP. So, the trend was as expected

2. (a) Part 1

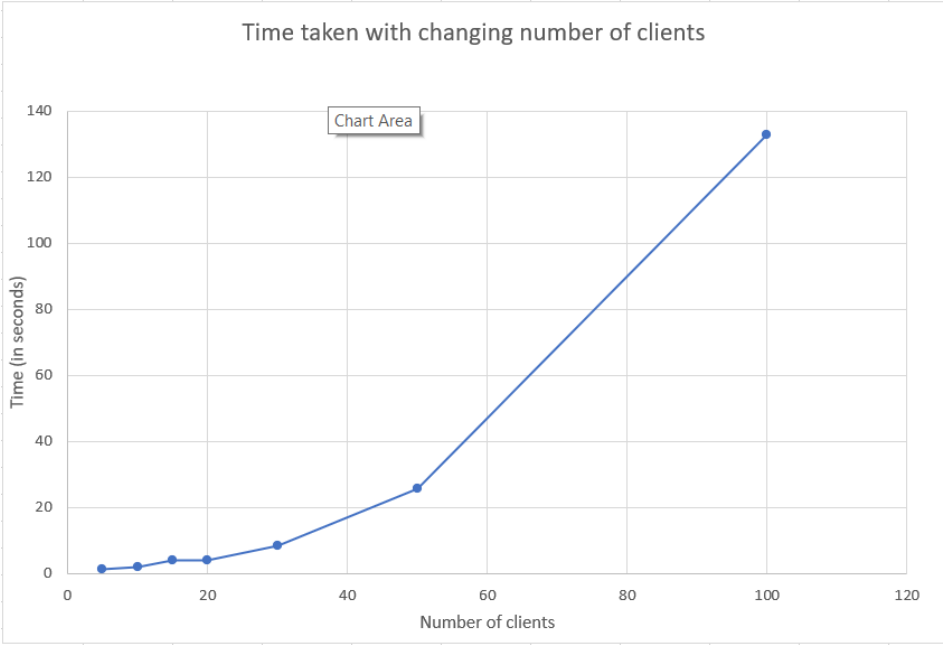
$\{1 : 0.0210774540, 2 : 0.0030527710, 3 : 0.0028627514, 4 : 0.0042929053, 5 : 0.0039840936, 6 : 0.0025003552, 7 : 0.0048609972, 8 : 0.004349589, 9 : 0.003168106, 10 : 0.00291901, 11 : 0.00282549, 12 : 0.00476956, 13 : 0.00366729, 14 : 0.00253170, 15 : 0.002653419, 16 : 0.00319367, 17 : 0.003104150, 18 : 0.00213503, 19 : 0.002345979, 20 : 0.003286778, 21 : 0.002525269, 22 : 0.00223284, 23 : 0.002361714, 24 : 0.002234458, 25 : 0.129141, 26 : 0.002130925, 27 : 0.00261425, 28 : 0.00524097, 29 : 0.00248855, 30 : 0.003564238, 31 : 0.003789007, 32 : 0.002733230, 33 : 0.002505540, 34 : 0.00205349, 35 : 0.002642095, 36 : 0.001727819, 37 : 0.00393569, 38 : 0.001761972, 39 : 0.002737045, 40 : 0.003742933, 41 : 0.00584167, 42 : 0.00358492, 43 : 0.002670407, 44 : 0.00265640, 45 : 0.003428995, 46 : 0.00249069, 47 : 0.00145173, 48 : 0.002431750, 49 : 0.001820027, 50 : 0.003940820, 51 : 0.002478241, 52 : 0.0046287, 53 : 0.00309419, 54 : 0.003701984, 55 : 0.0037460, 56 : 0.00157517, 57 : 0.003683984, 58 : 0.003213107, 59 : 0.00569415, 60 : 0.002117872, 61 : 0.00813609, 62 : 0.00354677, 63 : 0.004077076, 64 : 0.002866446, 65 : 0.00353115, 66 : 0.002457022, 67 : 0.00231426, 68 : 0.00470137, 69 : 0.00292462, 70 : 0.003201544, 71 : 0.003467798, 72 : 0.005511701, 73 : 0.00349396, 74 : 0.003181338, 75 : 0.00516223, 76 : 0.0092169, 77 : 0.00387024, 78 : 0.00669223, 79 : 0.004331588, 80 : 0.0050666, 81 : 0.00438243, 82 : 0.00865334, 83 : 0.00432467, 84 : 0.004233777, 85 : 0.005771398, 86 : 0.00405329, 87 : 0.006618, 88 : 0.00286436, 89 : 0.00398439, 90 : 0.00287872, 91 : 0.003660023, 92 : 0.00691652, 93 : 0.003595829, 94 : 0.00811439, 95 : 0.00644010, 96 : 0.00585955, 97 : 0.005608141, 98 : 0.00560086, 99 : 0.0058861, 100 : 0.00689673, 101 : 0.0083, 102 : 0.0053916, 103 : 0.00647896, 104 : 0.00575637, 105 : 0.00564289, 106 : 0.0047622, 107 : 0.00671458, 108 : 0.00685149, 109 : 0.00442099, 110 :$

0.00460296, 111 : 0.00667637, 112 : 0.00494354, 113 : 0.00530940, 114 : 0.0053069, 115 : 0.00292485, 116 : 0.00238049}

(b) Part 2
{1 : 0.00669807195, 2 : 0.0028411746, 3 : 0.00037515163, 4 : 0.00117842675, 5 : 0.0017759799, 6 : 0.000124931335, 7 : 0.0023524165, 8 : 0.00089550018, 9 : 0.00110042095, 10 : 0.00091463327, 11 : 0.00118857622, 12 : 0.00024545192, 13 : 0.00034347212, 14 : 0.00334084033, 15 : 0.00156086683, 16 0.00145667791, 17 : 0.00114285945, 18 : 0.00112509727, 19 : 0.0010000467, 20 : 0.00024998188, 21 : 0.00034998188, 22 : 0.00024594185, 23 : 0.0008828188, 24 : 0.0077950954, 25 : 0.0034520030, 26 : 0.00101566314, 27 : 0.0018179416, 28 : 0.00063455104, 29 : 0.0024443268, 30 : 0.00137281417, 31 : 0.00132119655, 32 : 0.00134593248, 33 : 0.00111591815, 34 : 0.00112462043, 35 : 0.00255697965, 36 0.00197076797, 37 : 0.00146365165, 38 : 0.00147378444, 39 : 0.00143915414, 40 : 0.00147098302, 41 0.00032484531, 42 : 0.00157074526, 43 : 0.00177097320, 44 : 0.00163400173, 45 : 0.0054981112, 46 : 0.00236821174, 47 : 0.00123161077, 48 : 0.0014523267, 49 : 0.00215357542, 50 : 0.00213176012, 51 : 0.00088590383, 52 : 0.002175927, 53 : 0.0021838545, 54 : 0.00167661905, 55 : 0.00177097320, 56 : 0.00089907646, 57 : 0.00105619430, 58 : 0.0008937120, 59 : 0.00093138217, 60 : 0.00088024139, 61 : 0.0021540522, 62 : 0.002358257, 63 : 0.00136661529, 64 : 0.00175911188, 65 : 0.00052618980, 66 : 0.00024807453, 67 : 0.00197672843, 68 : 0.0026051402, 69 : 0.0036481022, 70 : 0.00155687332, 71 : 0.00106984376, 72 : 0.002654254, 73 : 0.00135201215, 74 : 0.00037503242, 75 : 0.00268214941, 76 : 0.0026519894, 77 : 0.00175833702, 78 : 0.00112652778, 79 : 0.00117784738, 80 : 0.00050026178, 81 : 0.00232708454, 82 : 0.00063699483, 83 : 0.00053299493, 84 : 0.00171297788, 85 : 0.00196462869, 86 0.0032590031, 87 : 0.00037449598, 88 : 0.00200748443, 89 : 0.0020242333, 90 : 0.00032240152, 91 : 0.0020372271, 92 : 0.00238209962, 93 : 0.00195044279, 94 : 0.0025074481, 95 : 0.000311076641, 96 : 0.00063729286, 97 : 0.00025105476, 98 : 0.0057201981, 99 : 0.00320333242, 100 : 0.00214409828, 101 0.00113731622, 102 : 0.0016424655, 103 : 0.00237494707, 104 : 0.00134891271, 105 : 0.00260764360, 0.00024974346, 107 : 0.00032484531, 108 : 0.0015522837, 109 : 0.0076478719, 110 : 0.0009040832, 11 0.00126916170, 112 : 0.00470024347, 113 : 0.00206673145, 114 : 0.00090456008, 115 : 0.0015522837, 116 : 0.00308233499}

In this the for Part 1 1st packet is having slightly higher average RTT. I believe this is due to initial distribution of chunks which happen and due to this initially all 4 clients except the 1st one request for the 1st packet. Other than this packet 25 is having higher RTT this is because this is the 1st chunk requested by client 1.
For part 2, all the packets nearly have similar RTT.

3. For n = 5 clients time taken = 1.997919559 seconds
n = 10 clients time taken = 2.0125081539 seconds
n = 15 clients time taken = 3.9548802375 seconds
n = 20 clients time taken = 3.9872374534 seconds
n = 30 clients time taken = 8.6268832683 seconds
n = 50 clients time taken = 25.9001212120 seconds
n = 100 clients time taken = 132.74341273 seconds




This is as expected because on increasing the number of clients, the number of chunks each client receives will be very low. Hence, time taken would be quite high as lot of exchange of packets would need to be involved.

4. My program didn't ran completely on the large file. It took a large amount of time and it was not terminating. This is due to the large number of chunks in which the large file gets broken into.

5. Theoretically, the sequential method would take longer time as this would mean that the client has to look over each port of the server to request and then send it. Whereas if we choose a good randomizing function then we can get a better running time.

# Food for Thought

1. Here the main advantage would be of the space used by the server because each time it has to create a copy of the original data file and send it completely to each client. Whereas in our case, after sending the initial chunks, server can delete the data of the file. Other than this we can also say p2p networks are faster and scalable.

2. The method used by us would be a more efficient method than the traditional p2p system, this is because the clients here do not need to process large amount, their role is essentially to ask and provide data to and from the server respectively. Whereas the traditional p2p case would be more robust and would require more computation taking place at each client.

3. The difference would be that we would need to attach a file number also with packet along with the packet number. Chunks could be equally distributed among clients, equal in terms of total packets present in all the files combined.Further request by the clients could be made random to choose any file and request that packet from the client. Clients would now need to send two numbers as requests, file number and the chunk number. The storage in my simulation would need to be a list of list of dictionaries to uniquely identify each client with its each file and a chunk within each file.