Computer Networks Assignment 2

Ayush Verma

September 20, 2022

Contents

1 Notes

1. In Part1, I have used TCP protocol for all those transmissions where the protocol is not specified in the assignment
2. I have implemented the LRU cache using OrderedDict collection of python
3. Format to execute the shell script

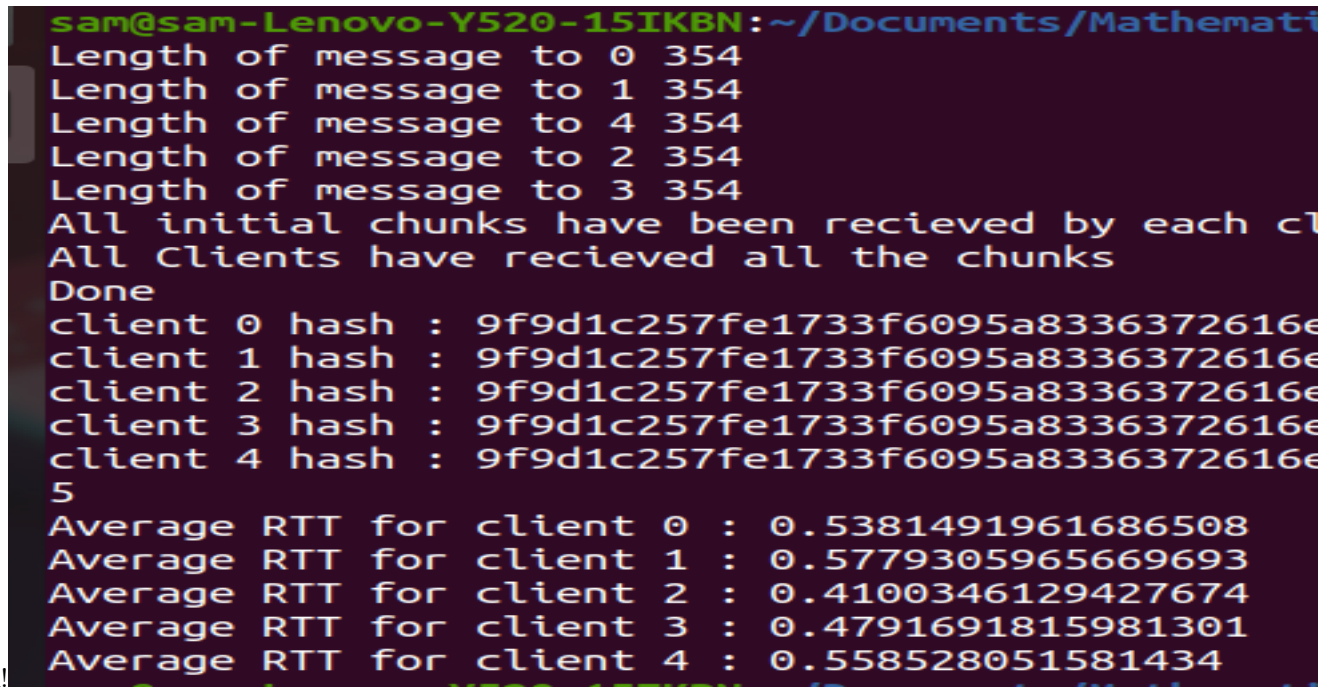
```
bash 2019MT60749.sh A2_small_file.txt
```

2 Analysis

1. **Record the RTT for each chunk a client requests. Report the average RTT over all chunks across all clients for both parts. Which RTT is higher? Was this expected? Why?**
   Ans. Each Request is for a chunk of size 1KB. The values are for n=5(number of clients)

Client	Average RTT for small file(in s)
0	0.3483043209366176
1	0.34258103370666504
2	0.3631419315133044
3	0.3654641310373942
4	0.3383958826782883

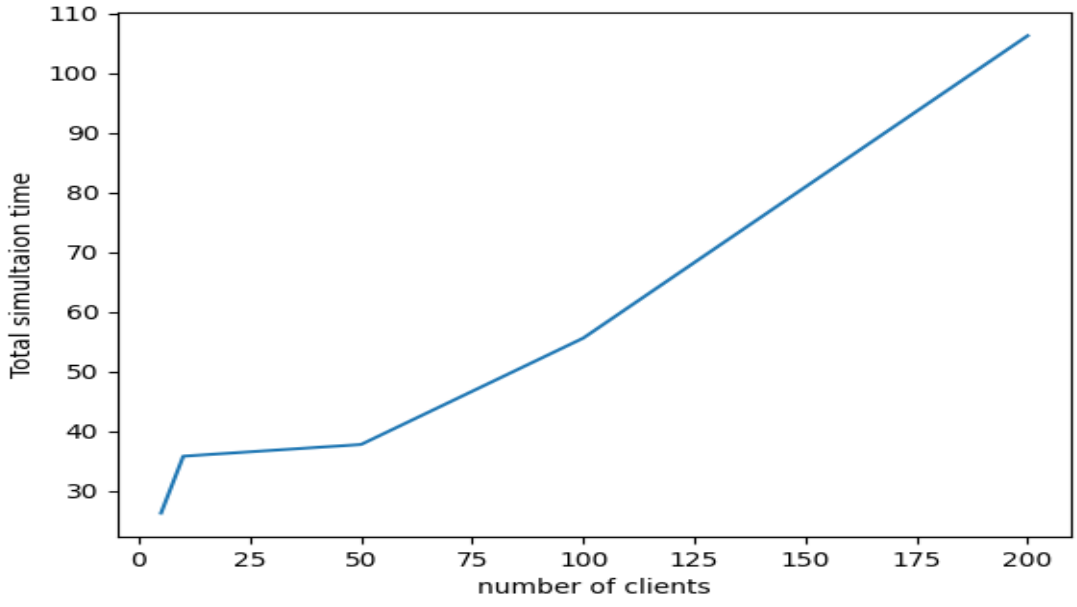
   Average RTT for all the clients is nearly same which was expected Total simulation time for n=5 with small file took :26.22601556777954



Ωcm!

2. **An important advantage of a traditional P2P network is scalability. By avoiding the presence of a central server, we go around creating a single node as bottleneck, as load increases. Let us experiment with the value of n. Run the simulations for n = 5, 10, 50 and 100. Plot the total time taken for the simulation, right from the beginning, vs n. What do you observe? Was this trend expected?**

number of Clients	Total simulation time for small file(in s)
5	26.22601556777954
10	35.76757454872131
50	37.72865319252014
100	55.56915235519409
200	106.27499628067017



Ωcm!

We can see the total simulation time increases as the number of clients increase. This is due to the too much overhead(requests) to the server. The server alone had to server all the requests for the chunk for

each client thorugh different ports. Also Since cache size also increase , this might help the server, but the increase in the number of clients domninate here. For n clients total file the server needs to transmit is n*F(F = size of file). Yeah this trend was expected for this network

3. **Further, experiment with the size of the cache at the server. Take n=100 for this part. Use file A2 large file.txt 1 (MD5 checksum: 89e57cef9c27f8b45cbb37f958dea193). Try values 100, 200, 300, ..., 1000. Plot the total simulation time against these values.**
Ans.

Number of Clients	Cache Size	Total simulation time for small(in s)	Total simulation time for file(in s)
4	1000	12.93967342376709	9002.431312531103318
100	1000	70.99912285804749(larger than previous with cache size smaller)	

4. **The request can be sequential or random when the client requests a chunk from the server. Report the time taken to receive all the chunks in both cases. Which method takes more time? Was this expected? Why?**

Type	Number of clients	Total simulation time for small file(in s)
Sequential	100	55.56915235519409
Sequential	200	106.27499628067017
Random	100	79.11560845375061
Random	200	35.76757454872131

From the table , we can see that randomization makes the time worse. It is worseining simulation here because of the design issue. In the sequential way , server has sent the chunk i to client i *modulo* n. Also I had break the loop when broadcasting request is served . So this will cause benefit in sequential way rather than random requests. For example if client 0 doesn't have chunk 1 , so it will request chunk 1 first(sequentially) to client 1(sequentially). Now since client 1 has chunk 1 , the request will be automatically served from the server at first try. However in case of random requests client 0 might request chunk 2 to the client 1 and thus server will fail to serve the request at first try.

In general randomization should be better, because client may not know the design of the application.

3 Food for Thought

(a) **We know that P2P networks are faster and scalable because communication happens directly among peers. Here we still have a server through which all communications have to go. Can you think of an advantage this network has over a traditional file distribution system in which a server sends a copy of the whole file each time a client requests it?**
Ans. The PSP network we have designed had a benefit that the server can delete the file once it had sent all the disjoint chunks to the clients. Therefore in this way it can save space.

One more benefit we can see is that chunks received to the clients will not get lost even if server gets down . So, whenever server is up , clients get request for the the remaining chunks.However in case of traditional file distribution if the server gets down , entire file needs to be resend again to the client.

(b) **Can you think of an advantage of this network over a traditional P2P network?**

Ans. In P2P , clients should also participate in uploading the data and serving requests. Also there need to have clever trading algorithm to efficiently shrae and receive data from its peer.
However in our PSP network, clients only need to serve the request to the server and it can solely rely on server for getting all chunks. In our PSP network we can also design clever algorithm so

that whenever client request a particular chunk to the server, then the server chooses only relevant clients to request for those chunks

(c) **How would your simulation change in case there are multiple files across the clients and each client wants one of those? How would you construct a chunk so that the file it belongs to can be identified?**

Ans . We can follow the similar approach, i.e distribute disjoints equal number of chunks to the clients for each file and then each clients can ask their remaining chunks to the server for each file. For distinguishing chunks among files, we can provide the chunk ids sequentially for each file. For Example suppose file 1 has 20 chunks so its chunk ids will be 0,1.....19, and if file 2 50 chunks , then its chunk ids will be 20,21.....69 and continue for other files. We can send the total number of chunks for each file to client to distinguish them between chunks of different files. Another approach for distinguishing chunks among different files could be to use two ids one for file and other for chunks. So if file 1 has 50 chunks its chunk ids will be (0,0), (0,1).....(0,49)