COL334 Assignment 2

PSP

Monu (2020CS50432)

1. Part 1 (Transfering Data through TCP)

   In this Part I handled all requests over UDP connection and transferred file chunks over TCP connection. As TCP has more RTT so this implementation takes more time as compared to part2.

   Sockets Defined Server Side:

   - Data Initialization on Port 20001 : A TCP socket for transferring file cunks initially. This is multi-client single socket where all clients make TCP connection to get chunks of files for initializing PSP file transfer.
   - From Port 9000 to 9000+N-1: N UDP sockets for hearing requests of N clients.(Here we connect one client to one socket so that traffic on all connections is same.) These sockets get requests from N clients for any packet p.

   Sockets Defined Client Side:

   - UDP Sockets From port 7000 to7000+N-1: One socket on each client for hearing request for packet p. If client has packet p then it will acknowledge by sending msg = ”1” .
   - TCP Socket From port 6000 to 6000+N-1: One socket on each client for transferring Data. When someone make request for packet then it will send data corresponding to requested packet p.
   - TCP Socket From 10000 to 10000+N-1: One socket on each client for receiving data from the server. Server sends data packets on this socket. From received packet it extract Packet Number and data , and store Data to its local memory . Packet’s Application Layer design which is received by this socket:

     packet=str(Packet Number)+”/”+data

2. Part 2(Transfering Data through UDP)

   Data transferred over UDP connection and all Requests are made over TCP connections.

   Sockets Defined Server Side:

   - Data Initialization on Port 20000 to 20000+N-1 : A UDP socket for transferring file cunks initially. This is multi-client socket where all clients make UDP connection to get chunks of files for initializing PSP file transfer.
   - From Port 21000 to 21000+N-1: N TCP sockets for hearing requests of N clients.(Here we connect one client to one socket so that traffic on all

connections is same.) These sockets get requests from N clients for any packet p.

Sockets Defined Client Side:

- UDP socket from port 7000 to 7000+N-1: One Socket on each client for Recieving Data from server.
- TCP socket from port 8000 to 8000+N-1: One Socket on each client where we can make request for packet p. If it has data for packet p then it will send acknowledgement message=”1” .
- UDP socket from port 9000 to 9000+N-1: One socket on each client to get data packets from the server. It extract data and Packet number from the Packet and store it in its local memory.

Handling Data Loss in UDP:

- Application Layer Packet Layout transferred initially during Data transfer initialization:

  packet=`packetNumber&start%end/total?data`

  Each Packet contains info about from which starting packet to which end packet client had to make request.
- Similarly Packet’s Application Layer design which is received by UDP data receiving socket:

  packet=str(Packet Number)+”/”+data

  This design will help the socket which part of file’s data it is getting in data.

3. Analysis For Part 1 and Part 2

Report for Both part is as following:

1. RTT for the part which transfer data over TCP connection should be higer in comparison to the part which transfer data over UDP. Because RTT for TCP > RTT for UDP. Observations are same as expected.

   RTT for TCP> RTT for UDP.

   ➔ For N=30, average RTT for **part1** ≈ **0.2 sec** and average RTT for **part2** ≈ **0.02 sec.** As *part 1* transfer data over TCP it should take more time in comparison to *part 2.*

   ➔ For N=5, average RTT for **part1** ≈ **0.014 sec** and average RTT for **part2** ≈ **0.0065 sec.** As *part 1* transfer data over TCP it should take more time in comparison to *part 2.*

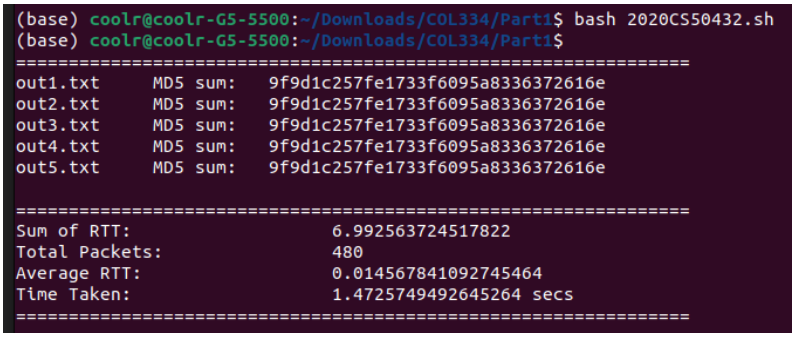


Fig: Part 1 Output for N=5 (RTT report)

```
(base) coolr@coolr-G5-5500:~/Downloads/COL334/Part2$ bash 2020CS50432.sh
(base) coolr@coolr-G5-5500:~/Downloads/COL334/Part2$
============================================================

out1.txt    MD5 sum:     9f9d1c257fe1733f6095a8336372616e
out2.txt    MD5 sum:     9f9d1c257fe1733f6095a8336372616e
out3.txt    MD5 sum:     9f9d1c257fe1733f6095a8336372616e
out4.txt    MD5 sum:     9f9d1c257fe1733f6095a8336372616e
out5.txt    MD5 sum:     9f9d1c257fe1733f6095a8336372616e

============================================================
Sum of RTT:             3.1241860389709473
Total Packets:          480
Average RTT:            0.006508720914522806
Time Taken:             1.2280945777893066 secs
============================================================
```

Fig : Part 2 Output (RTT Report)

2. Report the average RTT for each chunk across all clients, for both parts. Are there any chunks whose average RTT is significantly greater than the rest.
   The Average RTT for each chunk across all clients for both part is as bellow:
   For N=30, average RTT for **part1** ≈ **0.2 sec** and average RTT for **part2** ≈ **0.02 sec.**

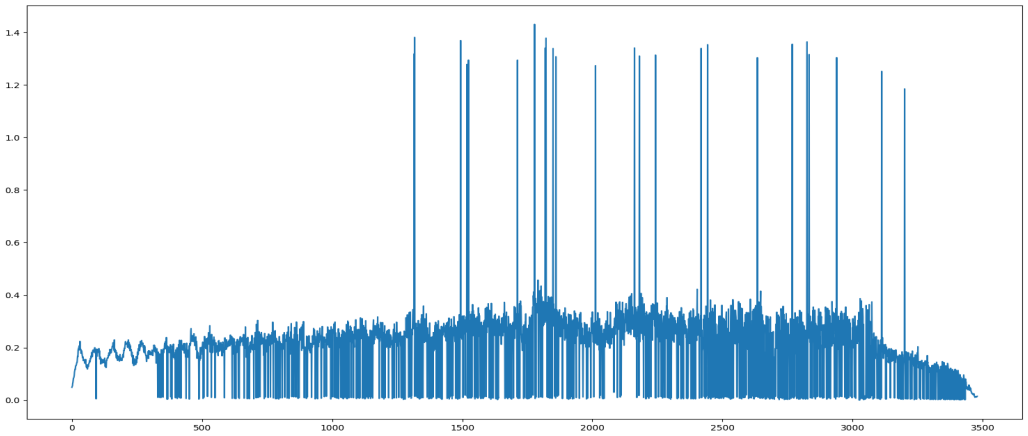


Fig 1 : RTT for all packets transferred for part 1 (N=30)

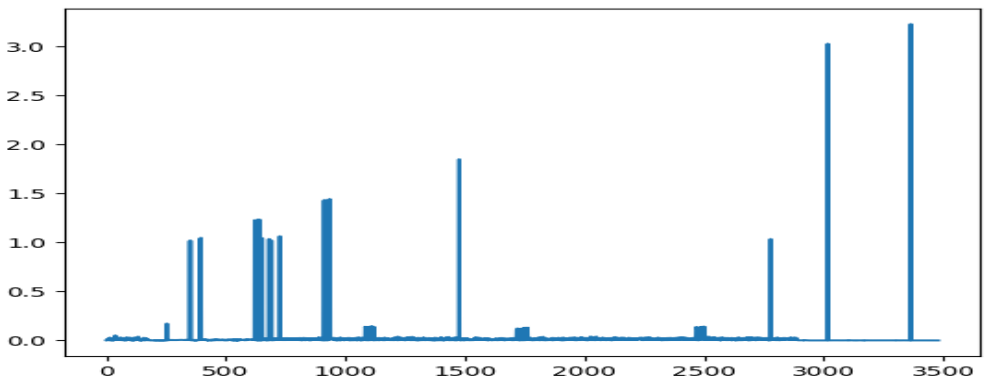


Fig 2: RTT for all packets transferred for part 2 (N=30)

As we can see from fig 1 there are some chunks whose rtt is significantly large in comparison to **0.2 secs** . Similarly in part 2 some packets have RTT very large in comparison to average RTT **0.02 sec.**

3. Plot for part 1 and 2 are as following:

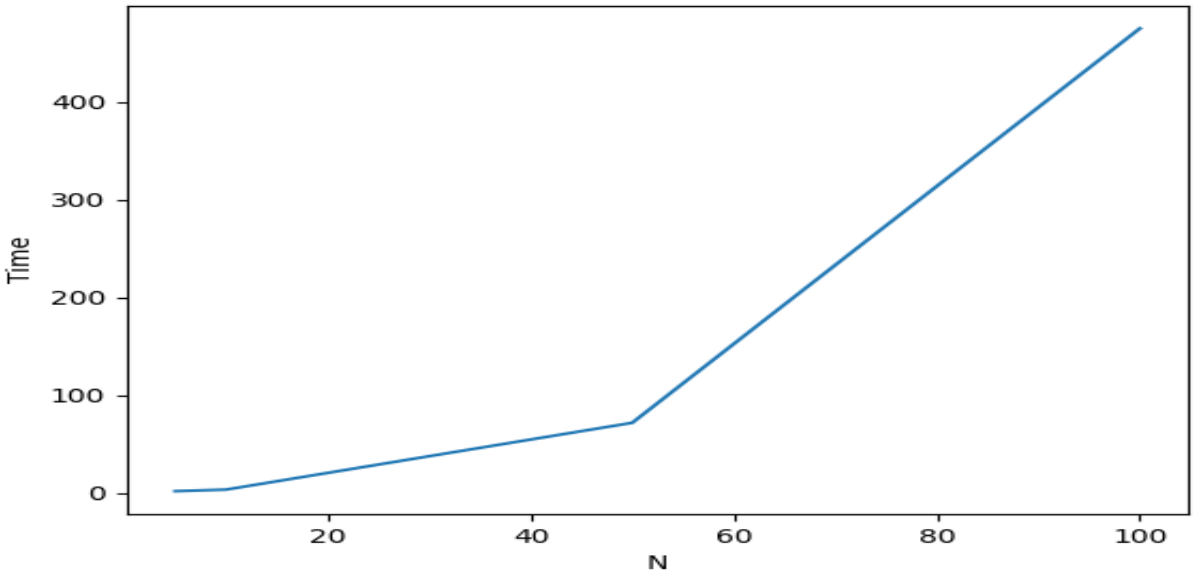


Fig 3: Plot for part 1

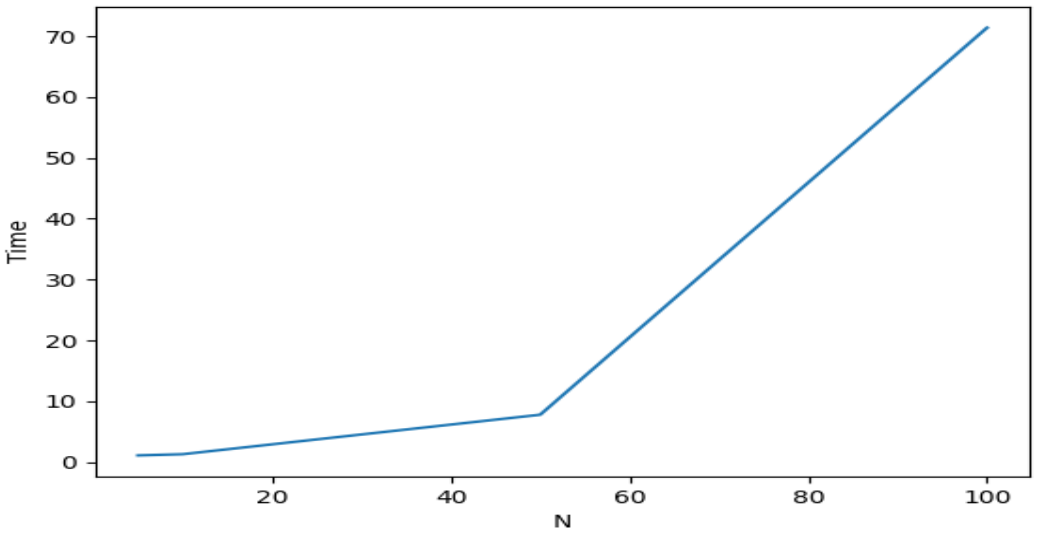
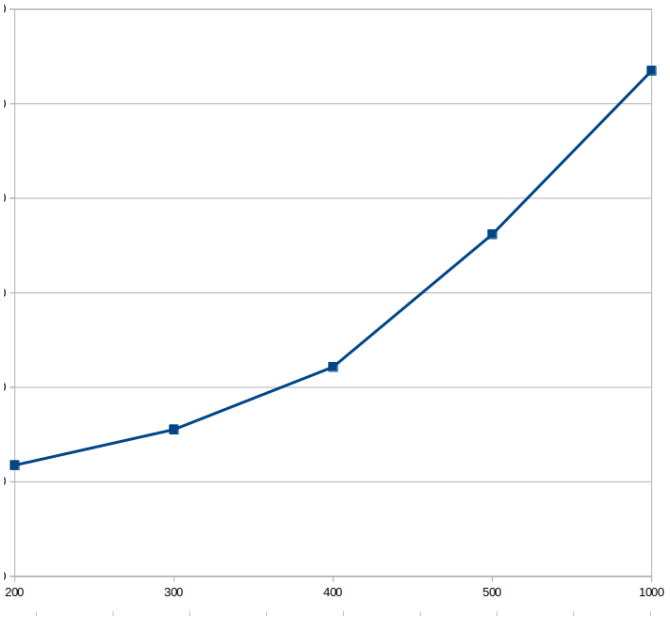


Fig 4: Plot for part 2

From Plots as shown above we can see Time increase as N increases.

N	Time Taken (Part 1)	Time Taken (Part 2)
5	1.56	1.09
10	3.15	1.29
50	71.47	7.78
100	474.91	71.40

4. Time taken for the process increase as we increase number of clients connected to the server. Plot for n=200,300,400,500,1000 is bellow:



5. With Sequencial Request we are getting better results as compared to Random Search. Time taken in Sequencial is less as compared to Random search. Yes, this is as expected because in sequential data requested from the client has more probability to hit in buffer. So, Sequential is better for our buffer implementation.

4. <u>Food for Thoughts</u>
   - This system will be faster as compared to traditional server to N clients file transfer for large N. Because as N increases number of service provider ports also increase which will make the file transfer process faster.
   - **Disadvantage:** Our implementation is very similar with PSP but maximum number of clients which can share file over the connection is less in PSP in comparison to P2P. Because we

have to make N ports on the server. But on P2P we don't need to have N sockets on server. So, Maximum clients which can share a file is much more larger as compared to our implementation.
Advantage: In P2P server don't have any option for updating it's cache in PSP we can update Server's cache with time. This is benefit of PSP over P2P.

- In our chunk we can add additional info about the file and sent it.We had used

  `packet=packetNumber&start%end/total?data`

  in our implementation. Now we require additional information that to which file chunk belongs. For this we make new updated chunk :
  `packet=filename:packetNumber&start%end/total?data` .
  This will help in identifying chunk's parent file.