COL374/672 Computer Networks: 2022-23 Semester I
Assignment 2
PSP Network

September 12, 2022

1 Introduction

In this assignment, you would build a network similar to a P2P file distribution framework. Traditional P2P, or peer-to-peer networks work with minimal to no reliance on a central server. Each connected node in the network, also called a "peer", is a user device: a laptop, PC, phone etc. These peers communicate by direct message passing instead of needing a shared server. A common P2P file sharing application is BitTorrent. Put in simple terms, peers in the network connect to other peers to request the missing parts of file they want. These peers also act as servicers by being open to sharing the parts of file they already possess.

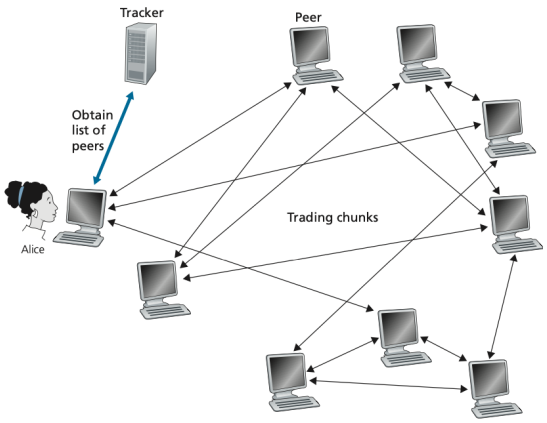

Figure 1: Simple P2P network

For this assignment, we would simplify the communication by having it through a central server. The server will act as a mediator to facilitate data transfer among peers. Hence the name PSP network. The assignment is divided into two parts. The parts differ in the protocol used for different types of communications.

2 Part 1

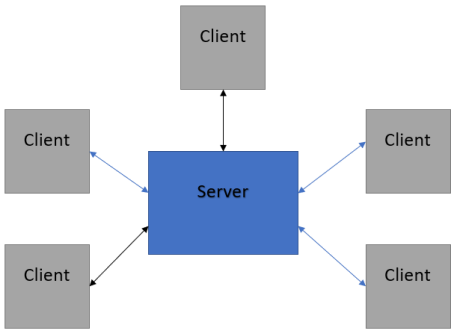

Figure 2: A simple PSP network with a server and 5 clients

The goal is to build a PSP network, that distributes the desired file to all the peers, starting with some distribution of file chunks among them. You are required to simulate the network following the steps given below:

1. Simulate a server and n clients (separate scripts have to be written for the simulation of server and client, in the client script, you can implement multiple clients using threads). Start with n=5.
2. Download the given file A2_small_file.txt 1. At t=0, the server should possess the complete file, and no client should have any part of it. The MD5 sum 2 of the file is 9f9d1c257fe1733f6095a8336372616e
3. The server divides the file into n disjoint but exhaustive chunks 3 and distributes one chunk to each client. At this point, each client should have one chunk, and no pair of clients should have any overlapping information. Alternatively, you can also use 1KB chunks for this task too. The parts of file shared still need to be equal in size, exhaustive and disjoint across clients.
4. Server deletes the file. Thus now, the server has no data with it.
5. In the process that follows, each client operates independently. From now on, each chunk of data exchanged would be of 1 Kilobytes. The simulation ends when each client has obtained all the chunks required to construct the file. The server 'S' has a cache installed, following LRU policy 4. This cache has a fixed size of n Kilobytes. S cannot store any other data outside this cache. Follow the process for each client:
   (a) Client 'p' identifies what chunk of file it doesn't have. Call it chunk 'c'.
   (b) p sends a UDP message to S, requesting c
   (c) S checks its cache for c. If the query results in a hit, it opens up a TCP connection with p and shares c with p
   (d) If the cache query results in a miss, the server sends a broadcast 5 to all clients, requesting c.
   (e) When a client receives a broadcast, it sends c to S through a TCP connection if it has it. If the client does not have the requested chunk, it ignores the broadcast.
   (f) S receives c from (possibly) multiple clients. It discards the duplicates, adds c to its cache, and sends it back to p via TCP.

(g) p, on receiving c, adds it to its available chunks. It checks if it now has the complete file. If not, it repeats steps 5.a-5.g.

6. Some more details:

   (a) Each client can be given access to ~~two~~ multiple ports. You are free to use any number of ports per client. Please justify the number you choose for each type of port in the report .

   (b) The server similarly can be given access to ~~n~~ multiple ports, to communicate with the clients concurrently. The server should be capable of having a total of n connections at once. So you can plan the number of ports you use accordingly. This again can be accomplished using threads. These ports can be fixed at the beginning of the simulation. Whenever a client wishes to request a chunk from the server, it would select a port uniformly at random and use that port only for that chunk. For the next chunk, a port will have to be selected randomly again. This will ensure that in the limiting case, no port at the server is disproportionately congested. Again, justify the number of UDP and TCP ports that you use, in the report.

   (c) Note that because communication via UDP is not reliable, you would have to handle cases of packet loss.

3 Part 2

In this part, the same simulation as Part 1 is repeated. However, now, all control messages should be carried out using TCP, and UDP should be used for all data transfer. Control messages here refer to all communications which don't include any part of the file to be shared. Examples would be client requesting a chunk from the server and the broadcast that server sends to all clients requesting a chunk. Any communication in which a chunk of the file is shared, comes under data transfer.

4 Analysis

Now, lets do some analysis based on these simulations. These will help you gain important insights into what is actually happening, and will encourage you to think why.

1. Record the RTT 6 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?

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?

3. 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?

4. 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.

5. 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?

5 Food For Thought

1. 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?
2. Can you think of an advantage of this network over a traditional P2P network?
3. 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?

6 Appendix

1. Both files needed for this assignment can be downloaded from here.
2. An MD5 hash is a widely used hashing algorithm which can encrypt a string to a 128-bit fingerprint. In this assignment, we would use it to check the integrity of the file received. More precisely, a client can ensure that it has received the whole file, and has arranged the chunks in the correct order by computing the MD5 sum of the file and comparing it with the given MD5 sum.
3. A "chunk" is typically a part of the file. There are different ways of dividing a file into chunks. In case of text files, a chunk can be one character, one word (tokenised by space), a sentence etc. With these techniques, the length of chunks can vary. A more robust method that can be followed is define a chunk by a byte range. Any file can be thought of as a long binary string. Then if chunk size is 2 bytes, then the first chunk of the file is simply the first 16 bits of that string. The next chunk are the bits 17-32 and so on. For this way of chunking, you would have to figure out how to access a given byte range in a given file/string.
4. LRU or least recently used policy evicts those elements from the cache which were needed to be read the least recently, irrespective of when they were added to the cache. When a new element, which was not in the cache earlier, is to be added, the cache needs to make room for it. It selects the element whose last access time is the earliest and evicts that. The cache can be thought of as a priority queue. An element 'a' accessed later in time than element 'b' would have a higher priority. The element with the least priority is evicted, if needed.
5. Broadcast here does NOT refer to the traditional broadcasts. You do not need to implement a broadcast algorithm to relay the message to all clients. The server can simply send a message to each client separately.
6. RTT refers to the round trip time of a packet. For this assignment, RTT would imply the time elapsed between the request that a client sends requesting a chunk, and the response it finally receives. This would be the RTT for that chunk.

7 Submission Instructions

You can choose to write the code in the language of your preference. However, please ensure that both server and client codes are written in the same language. Prepare a shell script which runs the whole simulation, and displays on terminal the checksums of files received by each client (only for n=5, part 1). It should also save the files received by the clients in the same directory. Strict plagiarism policies would apply so please ensure all code is authentic. Prepare a detailed observation and analysis report based on the points made above. Submit your report in PDF format with name as $\langle ENTRYNO \rangle.pdf$, the shell script as $\langle ENTRYNO \rangle.sh$ and your server and client codes named as $\langle ENTRYNO \rangle\_server.\langle ext \rangle$, $\langle ENTRYNO \rangle\_client.\langle ext \rangle$. Zip all these files into a single zip file $\langle ENTRYNO \rangle.zip$ and submit on moodle.