CS242 Final Project Phase 2 Design Document

Project Overview

The final goal of this project is to design and implement a fully functional Peer-to-Peer (P2P) file-sharing network. In this design document, we present our initial design of a best-effort, decentralized P2P file-sharing network that allows peers to upload and download files directly from each other without relying on a centralized server for data transfer and distribution. Each peer can independently join or leave the network. Basing some of our design on BitTorrent, we will address the basic bootstrapping process to allow peers to join the network, peer-to-peer discovery, and protocols for P2P direct connection.

Operational Definitions

Defining a peer:

The peers in the network are the clients that look to upload and download files through connecting with other peers to share information. The peers are not centrally owned, instead they are users connecting through their laptops, smartphones, etc. These pairs are intermittently connected to and communicate with each other through the network.

Defining neighbor:

Any host in the network that the client is able to establish connection with.

Defining tracker:

A tracker is a centralized server node that helps coordinate the communication between peers. It does so by storing the mapping of the peers (defined above) in the network to their respective IP addresses. It keeps this list of IP addresses of the peers within the network and gets updated whenever a peer joins or leaves the network. The peer will contact the tracker upon joining or leaving. The IP address of the tracker is fixed and programmed into our P2P application, such that when a new client launches, it automatically tries to connect to the IP address of the tracker to join the network as a peer. The tracker does not participate in file sharing and will not have any file data.

Bootstrapping process for peers joining the network

The bootstrapping process enables a new peer to join the network. The tracker is crucial in this process by providing an entry point for new peers to register and discover existing peers in the network.

<u>For new peer A joining the network,</u>

1. Registration with tracker
    a. The tracker will keep a list of peers that are currently in the network
2. Tracker randomly selects a subset of up to 10 peers (edited in our implementation for testing purposes) from the participating peers list and sends the IP addresses of this batch to newly joined peer A – they are now a's neighbors
3. A starts with requesting a certain file download from the first neighbor, downloading all the relevant chunks (less or equal to 3) for the requested file sequentially. For example, the terminal will prompt A for a file it wants to search for, A can enter "F1" to search for file 1, and if the neighbor has chunk 1, 2 and 3 of file 1, F1C1, F1C2 and F1C3 will be downloaded to A.
    a. If A's request is not fulfilled from the first neighbor, it goes through its own neighbor list and downloads all the relevant file chunks it needs for that requested file.
    b. If A has no neighbors, A can gain neighbors by fulfilling requests from requesting peers. In future implementation, we will allow A to request more neighbors from the tracker.
    c. If no one in the network close enough by 5 layers has the required chunk A needs, wait – Best effort!

** for more on peer discovery, refer to Method for peer to find each other on the network section

<u>For peer B who is already in the network and has a list of neighbors,</u>

1. B has already established persistent TCP connections with n neighbors
2. B keeps going through its list of neighbors to request to download the chunks it needs
3. B could gain or lose neighbors
    a. If B got assigned to another peer as their neighbor, that peer will be added to B's neighbor list when the TCP connection between that peer and B gets

successfully established and B receives a request from that peer for file download in our current implementation.

b. If B has no neighbors (they all left) or no neighbor has the required chunk, the user will have to re-enter the file requested at the prompt in the terminal, and if there's a new peer in the network, our Breadth-first-search will forward this request on behalf of the peer by sharing its neighbor lists so it can make new connections (Best effort protocol!). B can also gain neighbors by sharing files that other peers request from it.

For peer C who has gathered all the required chunks,

1. C checks whether it has all the chunks – if yes
    a. All peers will check whether it has all the 3 chunks of the file they are searching for (all files will be broken down into 3 chunks automatically) everytime it downloads something new.
2. C can choose to stay in the network and become another peer's neighbor when new connections establish, or leave the network and all its neighbors will know C has left when their connections fail (this is the expected behavior – again, best effort!). C doesn't need to do anything. In our implementation, we assume C will be altruistic and stay in the network.

Method for peers to find each other on the network

1. Each peer in the network will attempt to request chunks of the file they are searching for from all neighbors assigned by the tracker, one by one.
2. In the situation that your direct neighbors are unable to fulfill your requested file, you have the capability to find other peers on the network by contacting your neighbor's neighbors and deeper layers through our BFS algorithm. If no one close enough in the network has chunks you need, wait/do nothing.

Ensuring direct peer-to-peer connection

Peer-to-peer connections are direct in our P2P network because peers have direct TCP connections to each other to file/chunk share. When a joined peer communicates with peers in the network, the peer always creates a persistent TCP connection with our design. This way, the peers have a flowing open connection with other neighbor peers that doesn't need to be reestablished(reducing 3-way-handshake steps). In the case that a peer receives more chunks than it had before, its peers could get the new chunks as well without reopening a new TCP connection.

Fault Tolerance

- We will be employing a best effort protocol.
- *A peer joins a currently peerless network*
    - For the first peer in the network, we can only wait for more peers to join.
- *A peer is not able to retrieve or contribute more information from the network*
    - For any peer, if nothing can be done in terms of uploading new chunks and downloading chunks from its nearby network(5 layers deep), we will wait as a best effort protocol. This peer can retry file requests, as time can allow the network to propagate a requested file closer to the vicinity of this peer, allowing for the request to be fulfilled.

User Interface and Interaction

Any peer will be able to receive updates, clear instructions, and feedback about the state of the network and its relation to the network through the command line in the terminal.

- If no peers are currently in the network, our terminal will print *"Assigned neighbors: None. (You are the first in the network!)"* to notify the user.
- In order to join the network, the user has to run the file and as a command line argument, enter the port the peer process is running on. When a new peer joins a network, we can alert the user that we have registered it and assigned peers accordingly. The terminal provides a response from the tracker for its registered neighbors (e.g. *"REGISTERED NEIGHBORS: 10.155.8.185:8001"*), and then the peer process confirms its neighbors with a message (e.g. "Assigned neighbors: [10.155.8.185:8001]") and tells the user what chunks of the file(s) it currently "has stored." Upon the event where a peer needs a file that is currently in the network, the terminal will re-prompt entering the file name and the user will have to wait until someone in the network has the file (Best effort approach!)
    - When a peer downloads a file from a peer, it will relay its host and IP information so that the servicing peer can add a new neighbor to its neighbor list. This interaction prints an "ADD FRIEND" message across their connection with the peer's host and ip information.
    - Upon entering the network, we print to the user its initial list of neighboring peers, but we do not reprint this information after this point because it will be constantly updating and adding friends as it searches through new neighboring lists during BFS if its neighbors don't have the information it

needs. We just keep the user notified if files are found or not and reprompt them to retry if not found.

- The user will be prompted to enter the file they are trying to search for, e.g., F1 or F2 for file 1 or file 2. Clear messages will indicate whether any chunk of the desired file is found or downloaded.
- Upon receiving certain chunks of information, we notify the user of the file chunk status in a format as such: “Downloaded chunks: [F1C1 F1C3] Missing chunks: F1C2”

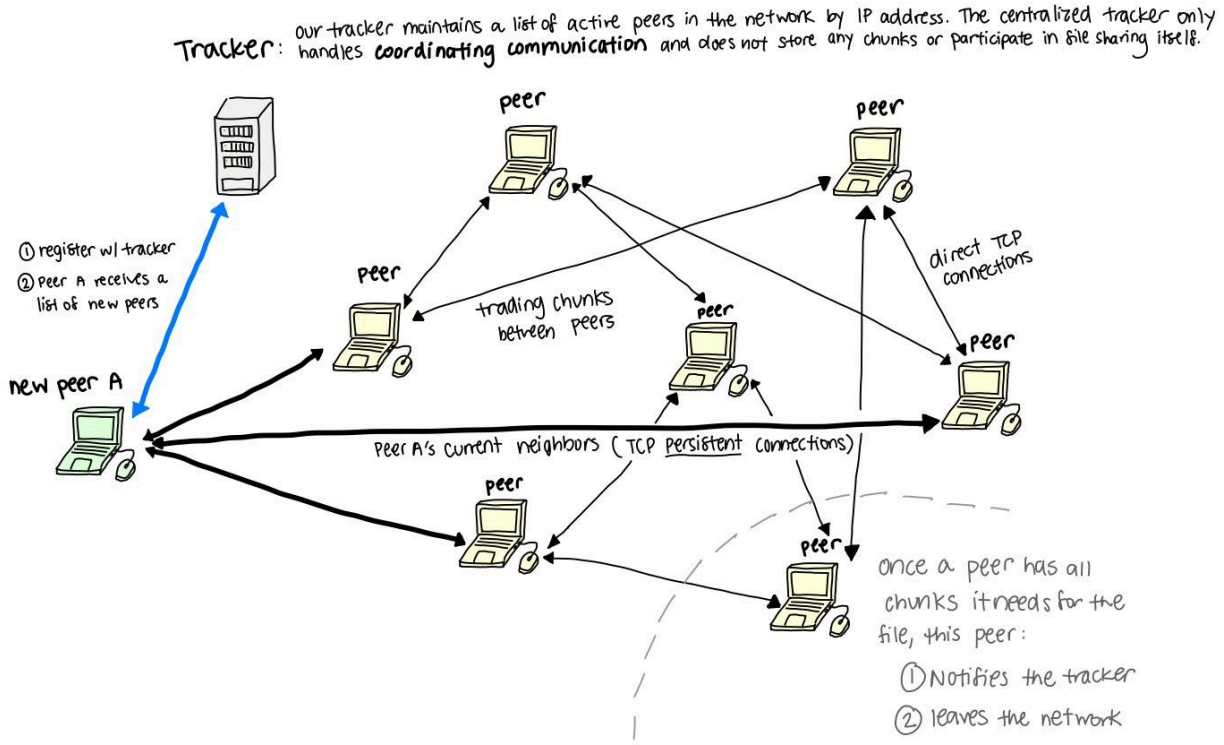
Such aspects of the user interface will allow the user to have transparency on its status and the status of files found in the P2P network.

Important components:

- Every peer would keep getting and losing peers
- Every peer keeps track of their own neighbor list
- Tracker keeps track of who’s in the network and is responsible for assigning new peers a random set of new neighbors

Helpful Diagrams/Visuals

Diagram: File distribution of our P2P network:



Phase 2 Team Reflection

Kitty

I focused on brainstorming with team members during our meetings, and analyzing the feasibility and pros and cons of various ideas. When we split work individually, I worked on refining our section about the method peers use to find each other on the network. I feel that the contact time we shared was a good amount and that we have very compatible schedules and ways of communicating for work.

Jessica

I worked on a brain dump for our initial meeting, and helped explore what different cases could happen and how we could address them. I refined the sections regarding the peer-to-peer direct communication, fallbacks, and user interface/interactions. I also made a more detailed version of the file distribution visual diagram. We split the work really well and all carried out our responsibilities well. I thought it was nice how we were on top of our communication and utilized office hours to gather more information and share with the group. It is very very pleasant working with Becky and Kitty.

Becky

I also contributed to the brainstorming and design during the meetings, and came up with our main direction with the team noting down the steps of our P2P design. When we split work individually, I focused on reorganizing the design document and write-up for the bootstrapping process section. I feel like our teamwork was efficient and productive.

Team

We met 3 times on Zoom over the span of a week as a team, totaling up to around 4.5 hours of active teamwork time. The majority of brainstorming and designing took place together. We also divided tasks for independent work, such as polishing the write-up, making visual components, and going to office hours for questions to improve efficiency. We all agreed upon the design direction and workload, and we were able to discuss ideas respectfully and preparedly. Our communication was done through Zoom, text, google doc comments, and Google Calendar.

CS242 Final Project Phase 3 - File Search Protocol Design Document

Start developing a protocol/strategy for a client to send a request for a file into the network and the corresponding result/reply to that request. Some of the questions to answer here are:

File Management
What types of files to use? • We will use text files (.txt) • 3 chunks of each file will be named in this format: F1C1 F1C2 F1C3 Will you keep each file intact or break it into chunks? If broken into chunks, how big are the chunks? • We plan to break the file up into 3 chunks for each file regardless of the size to maintain the requirement of keeping the network peer-to-peer. • This way we always know how many chunks a peer needs to download to complete the desired file they are searching for (makes testing easier as well) How will you distribute the files among the different peers as the network is formed? Note that it doesn't make sense for all peers to have all the files. • Peers in the network are constantly downloading AND uploading chunks of an entire file, so we start off our program simulating that each peer has at least 1 file. Thus, we plan to distribute the files in chunks in our p2p network based on the premise that eventually, some peer will enter with the complete three chunks to some file that other peers need. ◦ All possible files will be hard-coded and stored in a shared local folder for the simplicity and scale of this project. We will determine what files a peer enters the network with by randomly assigning a subset of all the total files to a peer when they enter. The peer stores the list of the files(filenames) they enter within their own peer information. This simulates a new peer entering with their own files. When a peer joins, we also create a local "upload" folder in their unique peer directory to represent what they would come in with, in a non-simulated environment. This folder stays empty in our implementation, but serves as a representation of what a peer would enter with. ◦ As the network is formed, these peers will come in with a random set of file(s), each already broken into 3 chunks, that can to be shared and searched for through the network, and that constitutes what they can upload (aka access from the local folder with all available files).

- When a peer downloads a chunk of the desired file from another peer, that file will be added to a local folder named after the peer's name (port and IP) that holds the chunks this peer has downloaded. This is to mimic the behavior of different peers being on different machines.

User Interface

How will the user request a file?

- A user requests a file by manually typing in the terminal the specific file it is interested in
  - Our program prompts: "Enter the file name you want to search for (e.g., F1 or F2)", in the terminal, and the user will enter the title of the file, for example, "F1".
- A peer sends this request to all of its immediate neighbors over persistent TCP connections and the request/response model one neighbor at a time in our current implementation design.
  - The peer's client thread will exhaust its direct neighborlist as the first point of contact for the file they need. They do so by sending a message broadcast over the connection with their desired filename in a format "SEARCH FILE: <filename>". Their neighbors can then process this request and either respond with a "FOUND FILE: <chunks>" broadcast back to initiation the sending and downloading of the requested file, **OR** return a response that helps the original requesting source peer "forward" their request through the network themselves using a BFS methodology(>>To be discussed in the next section!).

Functionality

How will that request be disseminated into the network?

- In the case that a peer is unable to get the needed file from their direct neighbors, we will continue deepening the search further into the network by use of *indirect neighbors*.
- As mentioned previously, the direct neighbor's server thread will process a peers request by replying a broadcast message either that the file was found in format "FOUND FILE: <chunks>", or will give a different response which is formatted: "FILE NOT FOUND, SENDING NLIST: <their neighbor list>".
  - This neighbor list is relayed back to the original requesting source peer to hold onto.

    - Then, the source peer continues their search in the direct layer (its direct neighbors), accumulating more distant indirect neighbors through the neighbor list relaying that happens when a neighbor doesn't have the file.
    - If the whole layer can't service this request, the source peer now obtains a list of indirect neighbors the next layer down (its direct neighbor's neighbors!), from which it can request its needed file with this new list.
        - Note that this list is trimmed down to **only include unique new neighbors. Any repeated neighbors already contacted or already in the list aren't added duplicate times.** This allows the source peer to have a concise list for its next point of contact.
    - If this list of indirect neighbors is populated, we repeat the process allowing the source peer to reach out to these distant neighbors. This is done **recursively**, allowing the source peer to go layer by layer, deeper into the network.
        - We set a Time To Live TTL value of 5 for this breadth-first search-esque algorithm to allow the source peer to only go 5 layers deep.

How will a peer who has the file being searched for communicate that with the peer looking for the file?

- The peer with the searched-for chunk will need to communicate with the peer requesting the file.
    - Case 1: The peer with the needed file is direct neighbors with the peer searching for that file.
        - The peer needing the file requests its direct neighbor with the file, and this will be communicated by sending the chunks of this file over their TCP connection
    - Case 2: The peer with the chunk is not direct neighbors with the original requestor source peer.
        - With our BFS method that propagates the request to a distant peer, the original requestor source peer is actually able to get the distant peer's host:port combination and make a direct TCP connection to the peer with the file. This request is not put onto the responsibility of the file-obtaining peer's actual direct neighbors.
        - As long as the file exists in the hands of a peer that is within the TTL = 5 layers of connection in the network of peers from the source

peer, the source peer will make a direct connection with whoever has the file.

- **TLDR:** As long as the peer who has the file is within 5 layers of the source peer, it will receive a request from the source peer and can respond by sending back the chunks through the TCP connection once they are connected(best-effort).
- Note: We are assuming that the job of a peer is to reject or accept chunk requests based on if it has it, and it forwards those requests to its neighbors. The tracker only gives new assignments to peers with no *active* neighbors (e.g. new peers in the network).

Phase 3 Team Reflection

Kitty

For this phase, all our team members met together 3 times to discuss overall implementation decisions and troubleshooting our peer and client server. We split a lot more of the work together and I worked on bringing questions to office hours and sharing with the group and brainstorming ways of making our decisions work with Golang (especially data structures of our Peer). Because this phase required more work, we decided to meet in pairs to work together at least once for each pair. I was supposed to meet with Becky this week to code but couldn't this week, so I worked on implementation on my own.

Becky

Over the span of one and a half weeks, our team met multiple times on Zoom as a whole and met in pairs to tackle assigned tasks or discuss. We had this teamwork strategy in which we would always decide on a clear, general direction first and then all code together to lay out the structure of the functions we needed to consider and whether we had any questions. Then we would individually claim tasks and work on them until the next time we met, and this worked pretty well for our team. I mainly focused on coding the peerServer, filling out details in peerClient so that we could start testing, and organizing the peer registration logic in the tracker. I think we all contributed as much as we could and made pretty good progress as a team.

Jessica

In this phase we met multiple times to code in real-time altogether, bouncing ideas off each other as we wrote out the skeleton and general logic of the tracker. We also met in pairs to continue drafting ideas for the Peer Client and Server logic. I helped with planning out logic and writing individual code for downloading chunks and creating connections between the peer and its neighboring peers(hardcoded), as well as general brainstorming for the methods we needed. I also went to office hours to gather information and report back to the group with new ideas and findings! I also helped clean up, finalize, and polish our File Search Protocol to be clear and ready for submission. Becky put in a lot of work to put all our ideas together and rework logic + provide more code to bring the code together to work, which was really amazing. Kitty helped bring valuable perspective and when we brainstormed. It was great!

Team

In this phase, our team worked well together to mix real time peer programming and also individual code to share with the group. We also were able to really analyze how the code needed to be altered and was able to view our logic in a new light. Office hours helped us clarify things and also spark more critical thinking about how we could take the feedback and work it into our plan without completely erasing our progress. Discussing with the team together here was very crucial to our momentum to the end.

CS242 P2P Project Final Phase Team Reflection

As we are reflecting on the project, as an aside, we also brainstormed some technical improvements we could have made, if we had more time.

- Regarding the **performance** of our code, we think it works efficiently since we deliberately made a many design decisions together such as:
    - Make TCP connections between peers and its neighbor only under the need of finding a file
    - During BFS, only visit indirect neighbors based on need instead of keeping a growing list of peers to contact

Current limitations and future improvements for them include:
***Note: During the presentation, the downloading of files would occasionally download oddly(combining chunks). This bug is fixed in our code now :) yay!*

- Neighbor Lists for connected peers are only updated to include each other if they participated in file sharing, rather than adding each other as a neighbor when they make a TCP connection regardless of file transfer.
    - Improvement/Fix: Ensure that when a TCP connection is made between two peers, they will update both neighbor lists accordingly.
- If a peer has no neighbors for whatever reason(all their peers left or they are the first to join), we haven't gotten to implement their request to the tracker for more neighbors. The only way they can get more neighbors as of now is if they send a requested file to another neighbor(the above fix will also make this better,) as they will be assigned to other peers as a neighbor single-sidedly until request and upload interactions happen.
    - Improvement/Fix: Allow a peer to send a message over its TCP connection with the tracker which states that it would like more neighbors, which will do the trick.

Jessica
In the final phase, I mainly worked on the code. I helped turn our previously hardcoded demo into a more dynamic one by allowing each peer to write their own port that they would like to connect to, refined the host:port mechanism so that peers are able to connect to multiple peers at once, and create repositories for each unique peer. I also worked on combining the peerClient and peerServer code into one peer so that the peerClient and peerServer behaviors can be done using multithreading. One of my main tasks was also implementing the breadth-first-search approach so that peers can reach further into the network for the

files they need. For final touch ups, I helped clean up the code and add lots of comments/documentation for each method. Kitty and Becky were so so incredibly helpful when we discussed all our implementation strategies and debugged together for hours on end. Kitty provided lots of valuable insight into the designing of the methods, and Becky worked on so many of the pillars and foundations of our project coding the communication between entities and setting up much of the structure. It was so great to be able to work with them and make so much progress with our final demo!!

Kitty

For the final phase, I worked on redesigning our implementation (i.e. revisiting and editing phases 2 and 3) based on the feedback from our presentation and working a lot in tandem with Jessica and all together to create more user interface for our program. Initially, we had divvied up the work separately and I was focusing on how to create the interface and code for the separate client and server processes, but then me and Jessica decided to work together because we had similar ideas and troubleshooting problems that depended on each other which led to us combining our peer client and server into the same process with different threads for simplicity and ease of sharing information. I also focused my efforts mostly in testing and debugging the peer process and the file downloading based on Jessica's work in the BFS algorithm and Becky's file searching function, although a lot of our work together was troubleshooting and synchronous co-working as a team as well.

Becky

In this final phase, I focused on optimizing the user interaction to make it more manual and interactive, re-structuring, and the general flow of the file search feature. We also troubleshoot a lot together as a team to brainstorm how to carry out BFS, debug existing code, and make the decision to merge peerClient and peerServer (and Jessica put in a lot of effort executing this implementation). I also contributed to the editing of phase 2 and phase 3 design documents to better reflect our updated designs after receiving feedback at the pitch in class before Thanksgiving break.

Team

We worked great as a team, communicating, peer programming, and meeting frequently in pairs to solve different steps of the project. We made sure to meet on Zoom at least once a week and increased the frequency as we got closer to the final deadlines. Everyone actively participated, communicated, and contributed to what we have as the final product of our project.