# PEER-TO-PEER FILE SHARING SYSTEM

*Akash Kumar Singh (16807068)*
*Yogendra Singh (160829)*
*Adhip Garg (160042)*

**GOAL:** The project aims to develop an application through which any user running this application on a linux machine can share files with other linux users with the same application installed. Users can get a list of all the files and directories (music files, pictures, etc.) of the connected users that they are willing to share.

**IMPLEMENTATION:** Due to lack of knowledge about the subject of networking we have taken some help from the internet in order to get an idea on how we can implement it.

For now we have divided the project into coarse phases or milestones that we would like to achieve to have a proper development of the project and to ensure robustness of each component of the system.

1. Discover the other linux systems around.
2. Enable the visibility of the files and directories the user want to share.
3. Get the lists of the files and directories of the other connected users.
4. Able to download files from the other user.
5. Adding extra functionalities and build a GUI around it.
6. Further optimisations.

**TIMELINE:** For now the timeline is very loose, in a sense that we don't have any idea of the skill set required to build something like this

1. Week 1: Develop the functionality of discovering the users around.
2. Week 2: Enable the visibility of the files and directories the user want to share.
3. Week 3: Add the functionality to get the lists of the files and directories of the other connected users.
4. Week 4: Enable the feature to actually download the shared files.
5. Week 5: Build GUI around it.
6. Week 6: Further improvement and optimisation.

*(2nd Week Report) 24/01/2020*

**Intro to Socket Programming:** This week our aim was to first get some understanding of socket programming and then write the code to achieve this basic functionality. So, we have written a broadcaster which sends a beacon message periodically and a listener which is always listening for this message, in order to connect.

**Discover the neighbours:** Inorder for the machines to find each other we need to periodically broadcast UDP messages so that anyone in the vicinity can catch those and know our presence. So, for this there should be two programs running on a particular machines at the same time: **Server** and **Client**.

**Server:** Server must periodically broadcast packets indicating its presence. We use the UDP protocol and include the following information in the packets: Identifier, MAC address, IP address, Port. Using this information the client can then make a TCP connection with the server and start talking with each other.

**Client:** Passively listen for broadcast beacon messages from nearby nodes. Keep track of which neighbors are currently within range. Note that neighbors will disappear as well as appear. You will want to print out the current list of neighbors so you can see who they are, only printing when there is a change.

We have created a Github repo with the code for broadcast and listener.

**Next Step:** After we have found a neighbour our aim will be to establish a TCP connection and be able to see the list of shareable files/directories on the connected linux machine.

Response to the comments on report1:

1. For functionality we meant to say that we will be focusing first on using CLI (command line instructions) for testing and using the system but later on we will add a very basic GUI.
2. Second addition to the functionality that we would like to make a distributed file sharing network out of it, like BitTorrent.
3. Optimisation will be done as we go one, on the basis of performance. Upon some searching we found that by flooding and routing using some algorithms can make a performance difference, so we will be looking out for that.

# Study Report (3rd Week, 31/01/2020)

## Current framework:

- The initial plan was to build a simple peer to peer file sharing system in which people can share files with each other like a shared google drive folder.
- So, anyone having the application would broadcast a UDP packet with all the necessary details like IP address, MAC address, etc., so that other users in the proximity will know about his/her presence and can access his/her shared files and vice versa for another user.
- Then the person would have a TCP connection to transfer the files.
- It was not a distributed file sharing system since a particular file is owned by a user completely, not in fragments.

## Desired framework:

- So according to the suggestions provided in the report 2 comments, we can change it to have different use case: “Shared Co-authoring of papers/journals (same file and multiple authors collaborating, making changes and final updation when every participant agrees over the changes)" like Google Doc but distributed.
- Upon going through some research and checking out some existing distributed file systems such as BitTorrent, we can think of the following requirements needed in the platform:
    - For something like google docs, we need to know with whom we want to share the file. So, we have to build a system which can identify between owners of file and others. One way is that we know the names of the nodes that are trying to access the file, like in google docs we give email addresses for the people which can access the files. Second way, we know a unique identifier to the file which is shared between the owners only like the shared link in google doc.
    - How we are going to store the file? It has many solutions but the first thing that came to our mind was that of BitTorrent. So, there are basically two components: client and tracker. Client is interested in file transfer. Tracker runs on a central server which can track all the active nodes which has fragments of a file that a client wants.
    - In our case if we need to edit a shared file then to know where the fragments of file are stored in the distributed network. The file fragments should be stored in such a way that only the users that have access to the file can request it. So, some kind of encryption or user identification has to be done in order to do this.
    - Also the location of the file fragments needs to be figured out. One option is to do like the BitTorrent, to have a central server, other option can be to store the info

in trusted nodes whose churn rates are low. For robustness it should be copied over multiple nodes to be safe.

## Questions:

- Since the plan of action changed a bit, from simple file sharing platform to distributed file system to distributed google docs, we would like to know of any simple implementation done like source code or step by step guide. Otherwise it would take too much time to just get the hang of all the topic and programming that goes behind implementing distributed applications. Any help would be appreciated.
- One peculiar addition to the complexity is authentication. How should we make sure that only the owners get the file not others. Any protocol to suggest.
- Would there be any application of DHT in it?
- What are the things that we should look into? Any pointers.

# Study Report (4th Week, 09/02/2020)

- After the feedback of the 3rd report we examined the possibility of modifying our design that will have a real time file sharing option that means a file is editable and can be modified by some other users.
- For this we have come up with the idea that every user in the dfs has access to files and they can also modify that file but for the purpose of having limited users that can modify the file we will have two different lists.
- First list will have the IDs of the users which are only allowed to view the files and the second list will have the IDs of the users who can both read and write to the files.
- Any modification made by the users of the second list will go to the author/owner of the original file and after the approval of the owner original file will be finally modified. And the modified file given by the modifying user to others will get deleted from the dfs network.
- Since each user will have the file path for the original file after final modification every other user viewing after modification will view the modified file.
- For users to write first they have to add themselves to the list of the file which have the authorized users who can edit that file. For this the user has to send a request to the author/owner of the file to get the edit access. Once the request gets approved the user will be added to the authorized user list who can edit that file.
- Users will have the freedom to choose the mode of viewing that is if a file can be private(view only to some allowed(by the author of the file) users) or it is public(view to everyone in dfs).
- For private files again users have to request for view mode from the author and once the request gets approved user will be added to the view list of the file and then user can view the files.
- Each private file will have two lists, one for reading and another for writing users.
- Each public file will have one list for editing users.

## Questions

- Concurrency of the file in the dfs due to duplicacy of the file after editing will have a impact on the performance of the dfs.
- If possible can we get some insights into the code for this.

# Programming Report (6th Week, 28/02/2020)

- Implemented and tested a server that periodically sends UDP datagrams containing its IP address, port, and MAC address.
- Implemented and tested a UDP listener that always listens for UDP datagrams to be able to get information about the IP and port no. of the server.
- Implemented TCP client and server. The client can send commands over the TCP connection for the server for listing its directory structure.
- For the TCP client and server can only make a connection. Code for directory listing and data transfer needs to be added.
- The code for the whole thing is on GitHub.

## Further Plan:

- Upon testing the broadcaster and the listener, the listener is able to receive the packet but it's not able to decipher the content in the packet. The received is not showing the data sent by the broadcaster. So, the next plan is to obviously fix that.
- The current TCP server and client are very simplistic (only able to establish connection), next we have to implement functionalities to actually send command and get data from server to client.
- Next, when we have got this, then we can make it much more distributed and storing data on multiple machines and trying out requesting files from the network.

## Questions:

- During the testing of packet transfer between broadcaster and listener, we found that when connected to iitk-sec through WiFi there was no data transfer between broadcaster and listener but when tried with a router on a private network it was working perfectly fine. So, can you explain what might be the reason behind this?
- If possible can we get some insights into the code for this? A starting point that we can use as a reference and build from that.