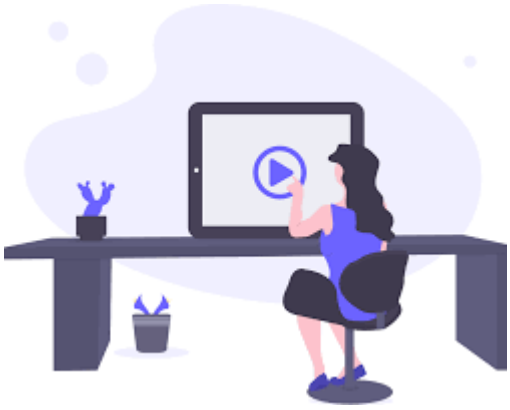
Vid - Streamer and Database Manager

...A peer to peer Video Streaming with a Distributed Database using MySQL and Firebase

Course Project for Database Systems and Computer Networks
Trimester 2 (2020-21)



Submitted to :

Dr. Romi Banerjee and Dr. Ravi Bhandari

Submitted by:

Aditya Kumar	Devin Garg	Kartik Vyas
[B18CSE002]	[B18CSE011]	[B18CSE020]

Contents

1 Introduction

Videos whether in the forms of movies, songs, informative videos, online classes, or any entertainment videos play a crucial part in the daily lives of a large group of the population. There has been a huge transformation in the mode of teaching especially in the past year because of Covid-19. Most of the classes are now being held online through different platforms. Now, there exist platforms such as Google Meet and Google Classroom, but these require the abundant need of data and the connection is not smooth unless Internet connectivity is very good. Thus, we developed our own system for real-life wide use cases. These cases include streaming a fun video among your friends, or an instructor taking a class and streaming the same to the students, or common file transfers of all kinds of files like videos, lecture notes, books, movies, etc. To implement a solution in this project, we have explored concepts around python sockets and different types of databases. The aim was to create an application that can be used for streaming videos via screen sharing across devices connected to a common network; the streaming video could be saved at the endpoints as well. Along with this, these videos would be saved in a backup folder which could be later accessed by the connected clients and transferred accordingly; for this, we have used MySQL[both, centralized and distributed databases] and an online Google Firebase database.

1.1 Motivation

As mentioned above, the main motivation behind this project was the need for a platform that can be used for video streaming and people can retrieve this data from the database later when needed. When coupled with the

content of this course, it seemed like a perfect overlap between a use-case and the concepts we were going to learn about in the Database Systems course. This application would ensure that over a network, all the connected endpoints can view the streamed video streamed by one server, and this video compiled with audio can be screen shared. These compiled audio and video files would be later saved in a backup for future sharing between these endpoints. The thing that was the most interesting in this project was how many problems it could solve simultaneously and the tech stack that we have used was quite exciting to learn and implement.

2 Problem Description

Briefly, it can be described as streaming of video via screen sharing to different devices connected to the same network and saving the streaming video in the database at the end of the stream so that it can be distributed to other clients anytime later. Considering the technical aspects, there would be one main server that will be streaming to multiple clients that would be connected to it and all these clients will have access to the saved video. Also, we will have a database backup folder associated with each user in their memory that can be used to save the recorded video depending upon the user's choice. Also, any user at any time can request data from other users through the help of sockets communication facilitated by the database.

3 Related work in the area

This problem has wide implications and usage in real life. Its importance is well understood in the market and there are many platforms like google Meet, Zoom in terms of video streaming and Google Drive, OneDrive in

terms of data sharing. Adding functionalities like screen sharing, saving recorded video over different devices connected in a network, and then backing up the database by MySQL and making it distributed and then alternatively using databases such as Firebase makes the project interesting and exciting. The apps that are currently available in the app stores are not open source and contributing to open source is motivating and with time, it will result in something that will be of usage for the masses.

The course content forms the perfect base for understanding the fundamentals of the project and the course helps in holding a grip and better understanding of concepts used in the project such as the connection of devices, their interaction by communication and file transfer, and then performing tasks in synchronization and maintaining a database and selectively performing file transfer. By using sockets and database tools fundamentally, the users do not need an internet connection to stream video and transfer the data. The work is highly scalable and is of ample use even in our own Institute where professors can stream the lectures from the quarters and students can attend classes from their rooms and further access the recorded lectures, just by connecting to the common network, an Internet connection is not required as such.

4 Background

Before diving into the details of our implementation in this project, let's briefly take a look at what sockets, MySQL database, and Google Firebase database are and how they function.

4.1 Sockets

Simply put, a socket is an endpoint communication instance for a node that is present in a network. In general, sockets include information about the transmission protocol in use, the IP address, and the port number. So, how

that works is - whenever a node acts as a server, it opens a socket and starts listening for connection requests. A client on the other hand is aware of the IP address of the server and the port on which the socket is open. With this information, the client is able to try to connect with the server, which, given some constraints are satisfied which may include authentication, then accepts the connection request, and then a connection is established between the client and the server.

4.2 MySQL Database

MySQL is an open-source relational database management system (RDBMS). A relational database organizes data into one or more data tables in which data types may be related to each other; these relations help structure the data. SQL is a language programmers use to create, modify and extract data from the relational database, as well as control user access to the database. In addition to relational databases and SQL, an RDBMS like MySQL works with an operating system to implement a relational database in a computer's storage system, manages users, allows for network access, and facilitates testing database integrity and creation of backups. MySQL has stand-alone clients that allow users to interact directly with a MySQL database using SQL, but more often MySQL is used with other programs to implement applications that need relational database capability.

4.3 Google Firebase Database

Firebase is a platform developed by Google for creating mobile and web applications. Firebase is a Backend-as-a-Service (Baas). It provides developers with a variety of tools and services to help them develop

quality apps, grow their user base, and earn profit. It is built on Google's infrastructure. Firebase is categorized as a NoSQL database program, which stores data in JSON-like documents.

5 Solution

This project can be broadly divided into six parts

5.1 Audio: Recording and Sending

We are recording audio from the speaker of the streaming device. We start two different threads. One for recording the speaker audio on the server-side, and the second thread to send speaker recording frame by frame over the socket. Similarly, we have two different threads on the client-side. One for receiving the speaker audio, and the second thread to play audio of the speaker. The audio stream is received on the client-side frame by frame and is processed further. The audio frames are appended to a file one by one. Both of the processes take place simultaneously. We are using the **pyaudio** and **wave library** for all of the above processes. At the end of the stream, the audio file is saved to the device and is merged with the video file.

5.2 Video: Recording and Sending

We are streaming video via screen sharing. We start with two different threads on the server-side. One for recording the video on the server-side, the second thread to send video frame by frame over the socket. Frames are compressed before being sent to the client. We are using **Zlib** and **pickle library** for compressing. Similarly, we have a separate thread for receiving the video stream on the client-side. We receive the video stream on the client-side frame by frame and it is processed further. Frames are decompressed and played and are appended to a file one by one. We are using **OpenCV, mss,** and **pygame library** for the above purposes. One caveat in the process is knowing the rate of the frames being transmitted. At the client's end (and at the server's end too), when a video stream is to be created the FPS at which the frames were recorded is needed. Hence, we ran a few tests and found that we could achieve a frame-rate of around 10 FPS in the best case.

5.2.1 Points of Learning and Hiccups

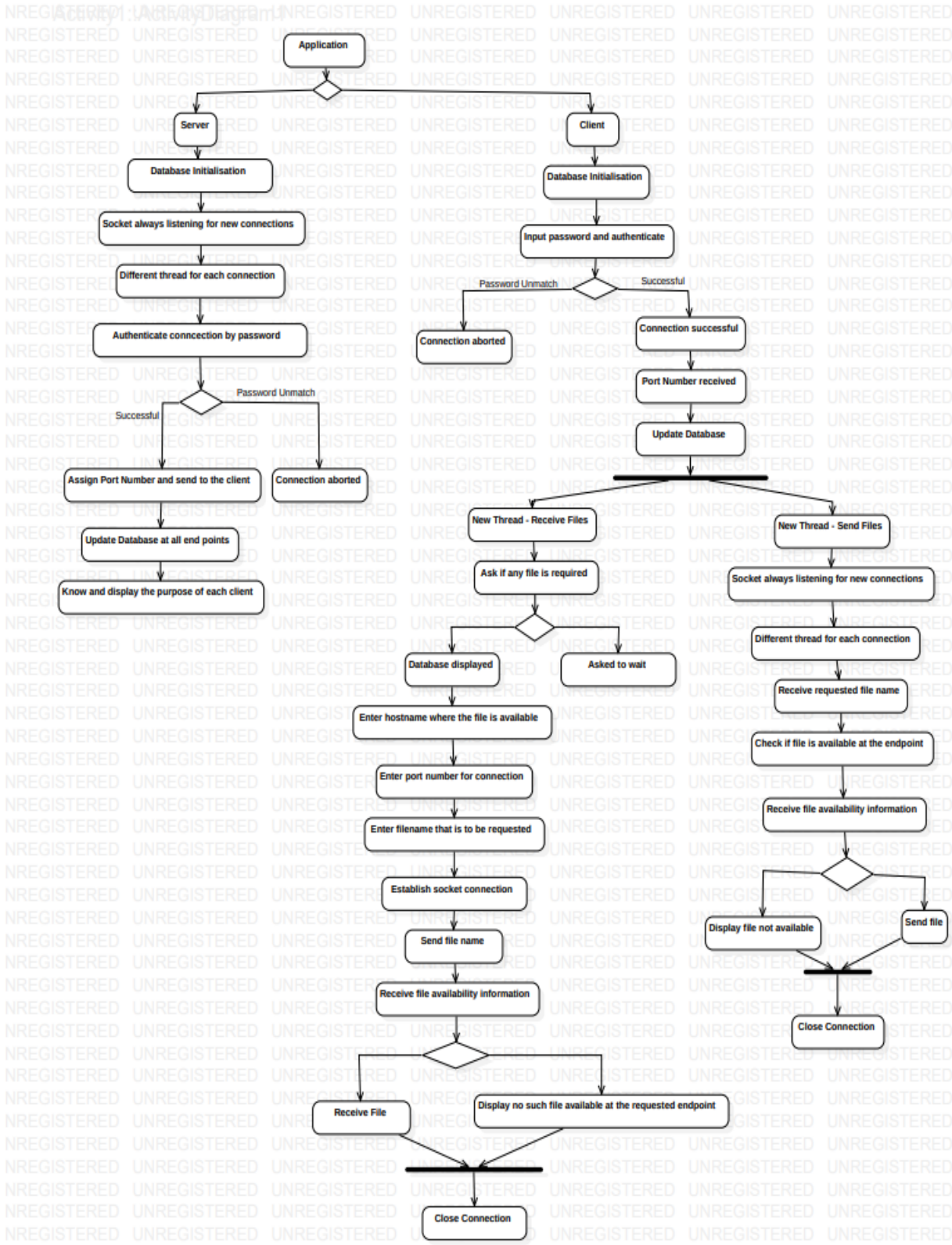
The major learning point in this segment (i.e. of video recording and streaming) was that of managing the corresponding frame rate. Originally, we were working with a frame rate of 30 fps; however, that was a mistake. If the actual frame rate of recording is not taken into account, the final video prepared is a lot more hastened. Therefore, we tried to find out at what rate the actual frame-recording was taking place and employed that for properly creating the final video stream.

5.3 Storing merged Audio and Video in Database

Once the user enters the stop command on the server-side, the streaming stops, and both audio and files and the video file are merged together. We are using the **FFmpeg** tool to mux the recorded audios and then to merge with the video together. Once the stream ends the recorded video is saved in the node for later use.

5.4 MySQL Centralised Database

The server would keep on listening for clients. The clients would connect to the server and then further enter the password; after proper authentication, the client can continue for further communication. Each client would update the database with all the files stored in their backup folder and then further each client would be able to see all the files that are available over the network. The database would consist of a single table having three attributes namely, file name, IP Address, and port number. Afterward, the client can get connected to the client that has the requested file via a socket and further file transfer can be facilitated. The database would be updated thereon and any client can join or leave the server at any point in time. The file transfer is very fast in nature as we are using direct peer to peer file transfer, for a transfer of a file of 1.1 GB, it took roughly 35 seconds. Below is the UML activity diagram for further ease in understanding the entire workflow of this approach :


Application
Server
Client
Database Initialisation
Database Initialisation
Socket always listening for new connections
Input password and authenticate
Different thread for each connection
Password Unmatch
Successful
Authenticate conncection by password
Connection aborted
Connection successful
Password Unmatch
Successful
Port Number received
Assign Port Number and send to the client
Connection aborted
Update Database
Update Database at all end points
New Thread - Receive Files
New Thread - Send Files
Know and display the purpose of each client
Ask if any file is required
Socket always listening for new connections
Database displayed
Asked to wait
Different thread for each connection
Receive requested file name
Enter hostname where the file is available
Check if file is available at the endpoint
Enter port number for connection
Receive file availability information
Enter filename that is to be requested
Establish socket connection
Display file not available
Send file
Send file name
Receive file availability information
Close Connection
Receive File
Display no such file available at the requested endpoint
Close Connection

5.4.1 Points of Learning and Hiccups in this Approach

The main thing to look out for was the functioning of the sockets and database individually and together as well. This led us to sharpen our concepts as we faced multiple problems while trying to implement the whole application.

Though any server that has MySQL workbench installed can become the server, but while the process is ongoing in case the server connection is interrupted, then the whole database data is lost, thus there is a chance of a single point of failure. Thus, we need to look for something, from where the database is not lost as soon as any endpoint connection over the network is interrupted.

5.5 Google Firebase Database

The socket communication and file transferring would be facilitated the same as above and for the database part instead of using a local database, we are using the Firebase database to remove the possibility of a single point of failure. Here, the database would be stored safely and there is no chance of database loss.

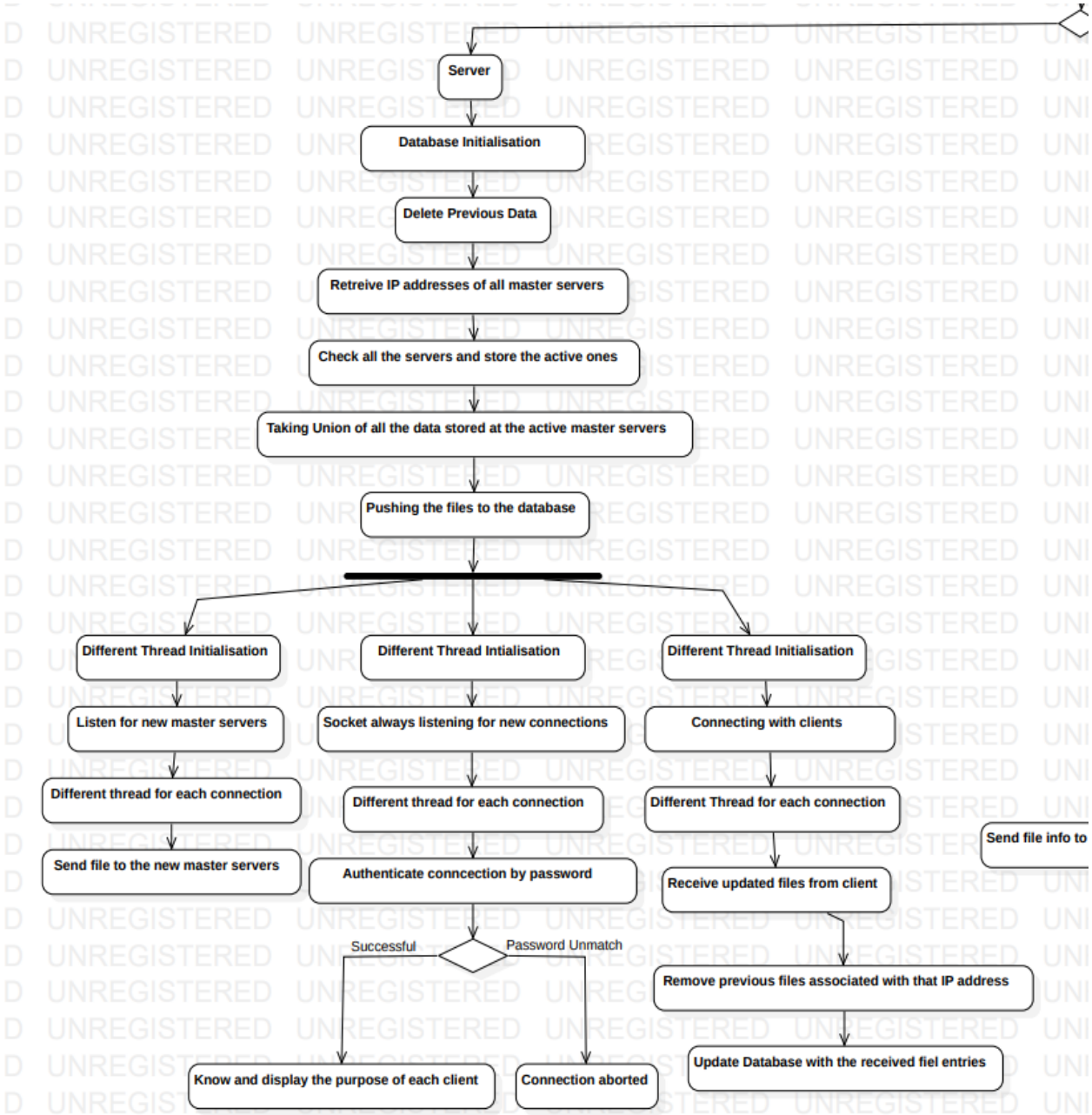
5.5.1 Points of Learning and Hiccups in this Approach

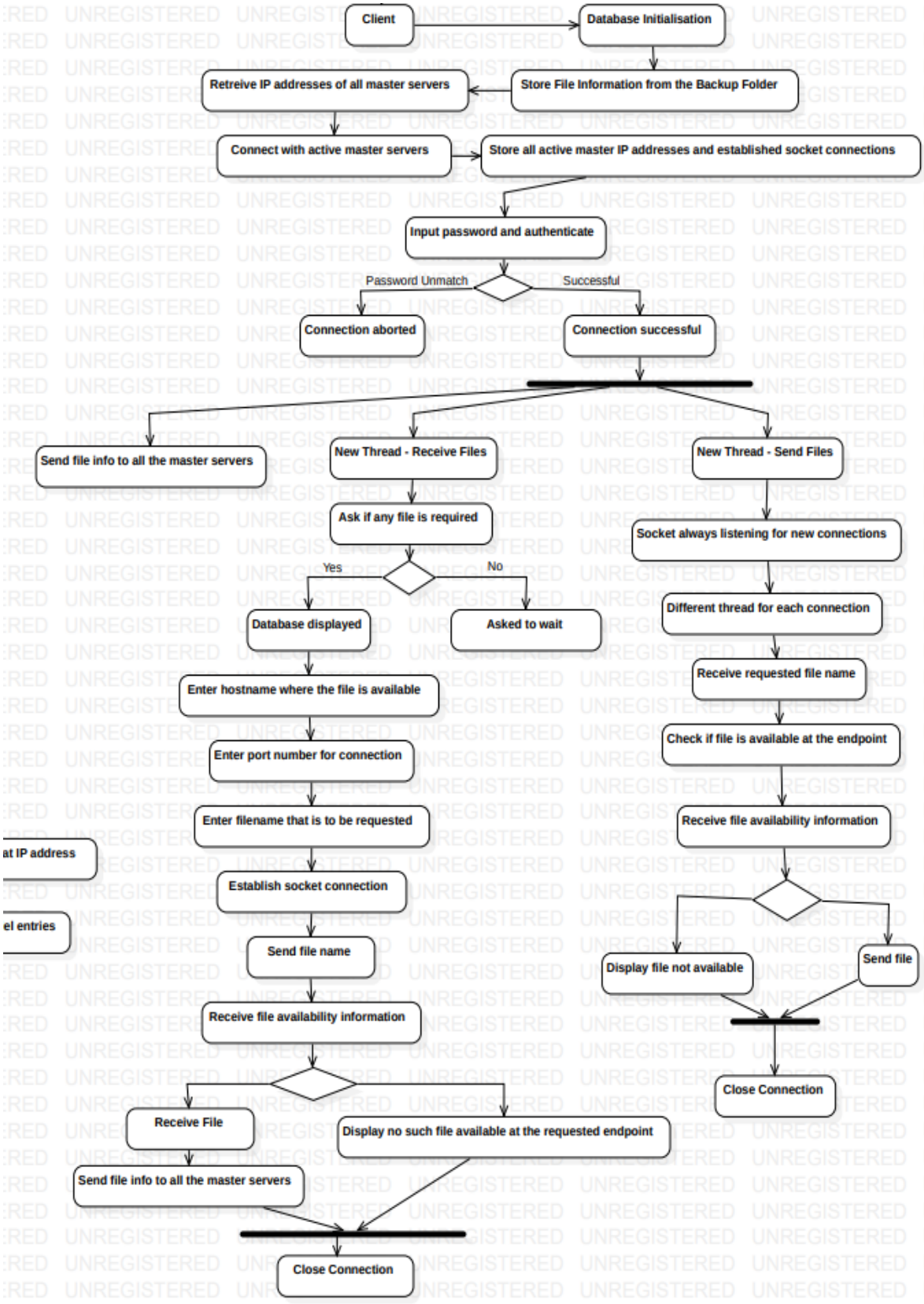
Learning and implementing an online database service like Firebase was really a nice experience and we got to see how to perform queries over such platforms and before that how to integrate online databases in our systems.

The only issue that exists is that for this we need an Internet connection, however since we have ensured that the data is just in form of a table, even loose connectivity would work in this approach. Next, we will look into how we integrated the MySQL database into a distributed database system.

5.6 MySQL Distributed Database

In this approach, we followed an approach of how the server and clients would structure the overall application. There would be multiple servers referred to as master servers and clients. All the master nodes will have a copy of the entire database, thus ensuring the smooth functioning of the application even when one connection of the server is interrupted. Once a master node has joined the network, it would retrieve the database from the already active master nodes. Further, whenever a client endpoint receives a new file, the database would be updated across all the master servers via a socket communication. Each client would be connected to each master node and the master nodes would be connected among themselves as well. Masters and clients have been designated specific port numbers in order to remove any unnecessary confusion and smooth functioning of the application. Below is the UML activity diagram for server and client sides for further ease in understanding the entire workflow of this approach :


Server
Database Initialisation
Delete Previous Data
Retreive IP addresses of all master servers
Check all the servers and store the active ones
Taking Union of all the data stored at the active master servers
Pushing the files to the database
Different Thread Initialisation
Different Thread Intialisation
Different Thread Initialisation
Listen for new master servers
Socket always listening for new connections
Connecting with clients
Different thread for each connection
Different thread for each connection
Different Thread for each connection
Send file info to
Send file to the new master servers
Authenticate conncection by password
Receive updated files from client
Successful
Password Unmatch
Remove previous files associated with that IP address
Update Database with the received fiel entries
Know and display the purpose of each client
Connection aborted


Client
Database Initialisation
Retreive IP addresses of all master servers
Store File Information from the Backup Folder
Connect with active master servers
Store all active master IP addresses and established socket connections
Input password and authenticate
Password Unmatch
Successful
Connection aborted
Connection successful
Send file info to all the master servers
New Thread - Receive Files
New Thread - Send Files
Ask if any file is required
Socket always listening for new connections
Yes
No
Database displayed
Asked to wait
Different thread for each connection
Receive requested file name
Enter hostname where the file is available
Check if file is available at the endpoint
Enter port number for connection
Receive file availability information
Enter filename that is to be requested
Establish socket connection
Send file name
Display file not available
Send file
Receive file availability information
Close Connection
Receive File
Display no such file available at the requested endpoint
Send file info to all the master servers
Close Connection

5.6.1 Points of Learning and Hiccups in this Approach

This approach was really complicated to implement as there are several sockets and multiple datasets that are involved. We tried to make the data consistent and reliable. The master servers and the clients interact with each other through multiple socket connections hosted at specific ports; we have used threading so that multiple processes can run concurrently.

6 Project Details

6.1 Codebase Link

https://github.com/garg-7/operation_overdrive

6.2 Demo Video Link

https://drive.google.com/drive/folders/1N7W310itR_toE0jzxt3ChLf0mQyirq_y?usp=sharing

6.3 Snapshots of the Demo

```
(env) PS D:\operation_overdrive\combined> python .\server.py
pygame 2.0.1 (SDL 2.0.14, Python 3.6.8)
Hello from the pygame community. https://www.pygame.org/contribute.html
192.168.29.158
Listening for connections...

********Input Devices***********
0 Microsoft Sound Mapper - Input
1 Microphone (Realtek(R) Audio)
2 Stereo Mix (Realtek(R) Audio)
*******************************

Mic Index #1
Recording...
Connected with ('192.168.29.158', 49978).
stop
screen share: 10.25 fps over 26.34 seconds
Stopped recording.
Saved server_speaker.wav.
Saved server_mic.wav.
```

Video Streaming Server

```
(env) PS D:\operation_overdrive\combined> python .\client.py
pygame 2.0.1 (SDL 2.0.14, Python 3.6.8)
Hello from the pygame community. https://www.pygame.org/contribute.html
Enter server IP192.168.29.158

********Output Devices***********
3 Microsoft Sound Mapper - Output
4 Speakers/Headphones (Realtek(R)
*******************************

Speaker Index: #4
Finished receiving audio stream.
Saved client_speaker.wav.
Saved client mic.wav.
```

Video Streaming Client

```
(env) PS D:\operation_overdrive\distributed\dbms and socket> python .\dbms.py
Timed out with 192.168.29.158

Timed out with 192.168.29.88

Listening for requests from other masters...
Enter the following address on the clients' ends for the socket connection :: 192.168.29.158
Connected with ('192.168.29.88', 54892)
Password authentication successful with ('192.168.29.88', 54892)
192.168.29.88 sharing update
Updating database
Database updated successfully
Connected with ('192.168.29.158', 51171)
Password authentication successful with ('192.168.29.158', 51171)
192.168.29.158 sharing update
Updating database
Database updated successfully
File Transfer Requested by address : ('192.168.29.88', 54892)
192.168.29.88 sharing update
Updating database
Database updated successfully
```

Database Backup Server

```
PS C:\Users\user\Downloads\Chrome\operation_overdrive-audio\n1> python .\client.py
Connected with master: 192.168.29.158
Enter the password to authenticate the connection:  letmepass
Received input :: Correct
Password authentication successful
Do you want to receive a file? (Y/N)Connected with master: 192.168.29.158

Y
Following is the file info of all the data available with associated host and port number
('192.168.29.88', 'b1.txt')
('192.168.29.158', 'b2.txt')
('192.168.29.158', 'back1.txt')
('192.168.29.158', 'back2.txt')
Enter the filename that you want alongwith the extension ::  b2.txt
Enter the hostname where the desired file is located ::  192.168.29.158
The requested file is getting transferred !!
File  transferred and saved in the backup folder with the same file name !
Connected with master: 192.168.29.158

Database Updated Successfully
```

Database Backup Client

7 Future Prospects

The wide real-life application makes this project's scalability pretty useful which can lead to meaningful end products. Possible future work that can be done is as follows

- Online Web Application with integrated chat
- Picture-in-picture webcam feed and screen share
- Adaptive FPS resolution of screen share at runtime

8 References

- https://medium.com/firebase-developers/what-is-firebase-the-complete-story-abridged-bcc730c5f2c0
- https://dzone.com/articles/sql-query-optimization-and
- https://www.youtube.com/watch?v=HXV3zeQKqGY
- https://www.c-sharpcorner.com/blogs/creating-local-database-using-microsoft-sql-server
- https://www.youtube.com/watch?v=1DhvKCjG2NE
- https://www.pygame.org/docs/
- https://people.csail.mit.edu/hubert/pyaudio/docs/
- https://stackoverflow.com/questions/48950962/screen-sharing-in-python
- https://www.geeksforgeeks.org/saving-a-video-using-opencv/
- https://stackoverflow.com/questions/30988033/sending-live-video-frame-over-network-in-python-opencv