



**School Of Electrical Engineering**

**And**

**Computer Science**

# **DISTRIBUTED COMPUTING**

## **SEMESTER PROJECT FINAL REPORT**

**PROJECT TITLE: SONIC SHARE**

NAME	CMS	CLASS
SYED AITEZAZ IMTIAZ	291093	BSCS-9C
UZAIR KHALID	295913	

**GITHUB LINK:** <https://github.com/aitezazshah/Project-Sonic-Share.git>

**SUBMISSION DATE:** 1/6/2022

**SUBMITTED TO:** Sir Shah Khalid

## Introduction:

Sonic Share is basically hybrid network designed to distribute any type of files in a fast and efficient manner. The important design philosophy was to not store the files in the central system but to store the list of files of each connect peer in the central indexing system. This is what basically a Peer to Peer file sharing system is. When a peer downloads a file, it will be basically downloading the file from another peer with the help of central index server.

## System Working and Architecture:

Overall system design follows the Napster's core architecture. It is hybrid network which means Client-Server + P2P style of communication. The system designed to work on hybrid structure which is evident that this is distributed System. A central indexing server accepts connection requests from peers and updates the data structure of the file name list (mapped to the peer) as soon as it receives from the peer. The server also allows the peer to search the file list. If the peer requests a file and its available with another active peer then, that peer's ID plus its location (port number) is sent to the peer. Now, the peer will connect to the peer holding the file and downloads it. Figure 1 depicts this idea.

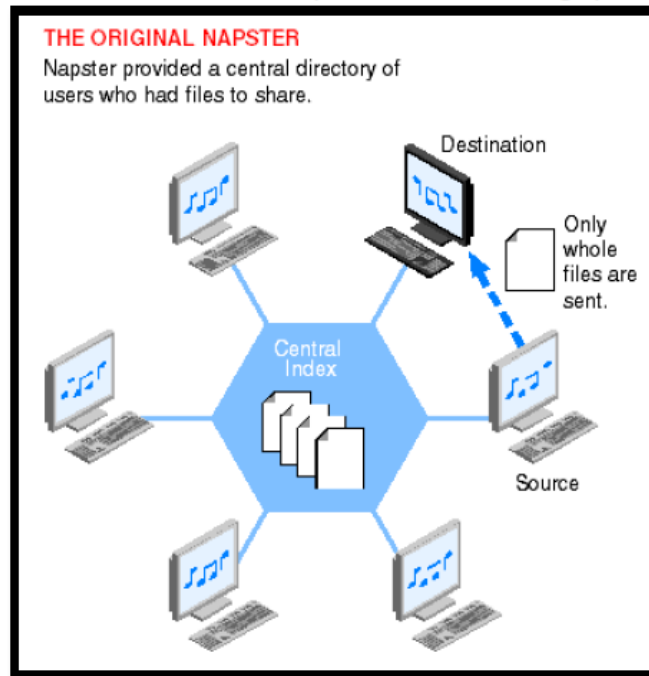


Figure 1 Napster's Architecture.

The communication between systems could be divided in 2 sub-parts:

**1. Indexing server** – Peer communication: Indexing server accepts requests and wait for the commands from them.

**a. REGISTER** : This command tells the server that peer is sending its file list. In the code design each file in the file list are sent one by one (to never overrun the buffer size) with REGISTER and peer ID (allotted by the server at the time of connection) as prefix. The format looks like this  
**REGISTER:<peerID>:<file\_name>.**

The Dictionary data structure is chosen to store file name list against the peerID. Basically, key is peerID and values are file name list. This key value pair are deleted as soon as peer becomes inactive

**b. SEARCH** : This command tells the server that peer wants a file. In the code design, file name is sent along with the SEARCH and peer ID as prefix. The format looks like this  
**SEARCH:<peerID>:<file\_name>.**

The Dictionary data structure is searched except for the peerID of peer demanding the file. If peers are found holding the file, its location (port number) and peerID are sent back to the peer

The other commands are *EXIT* to tell the server that the peer is going to be inactive. Any other commands are treated as illegal and that client is disconnected.

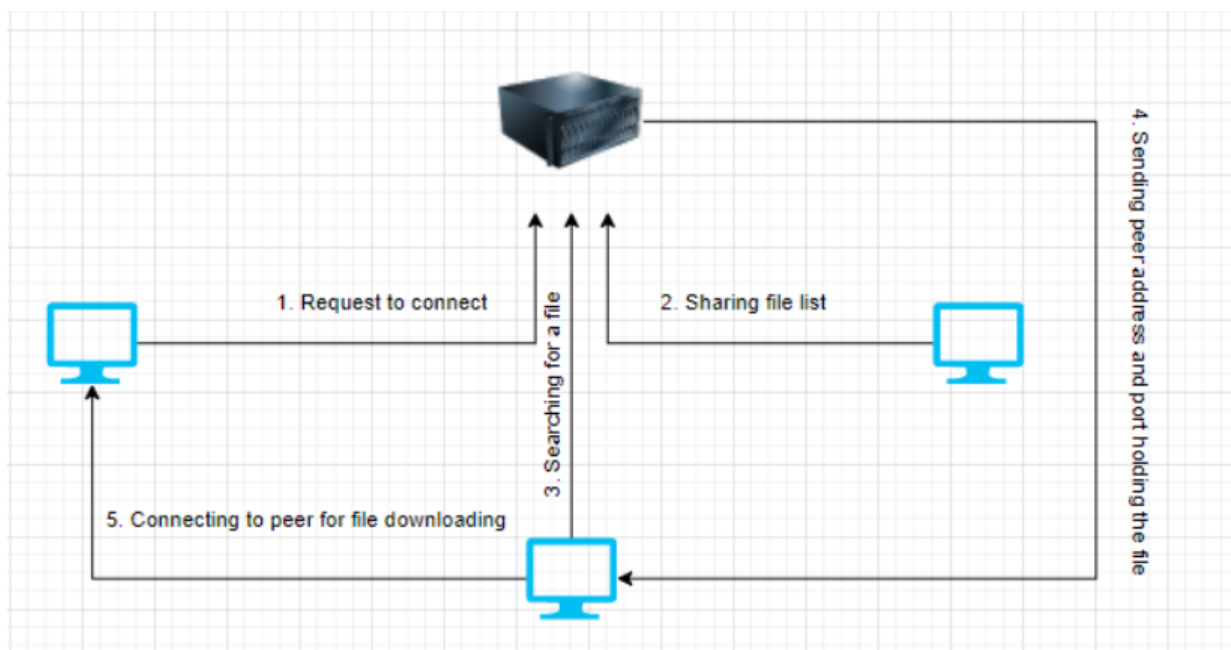
**2. Peer-to-Peer Communication:** Each peer is designed to be a server as well as client. When a peer is acting as server for a peer then, it's uploading the file requested by another peer. When a peer is acting as client to another peer then that peer is downloading the file.

When a peer accepts the request from another peer, it waits for the command namely, *OBTAIN*. A peer requesting the file will send file name with OBTAIN and peer ID as prefix. The format looks like this: *OBTAIN:<peerID>:<file\_name>*.

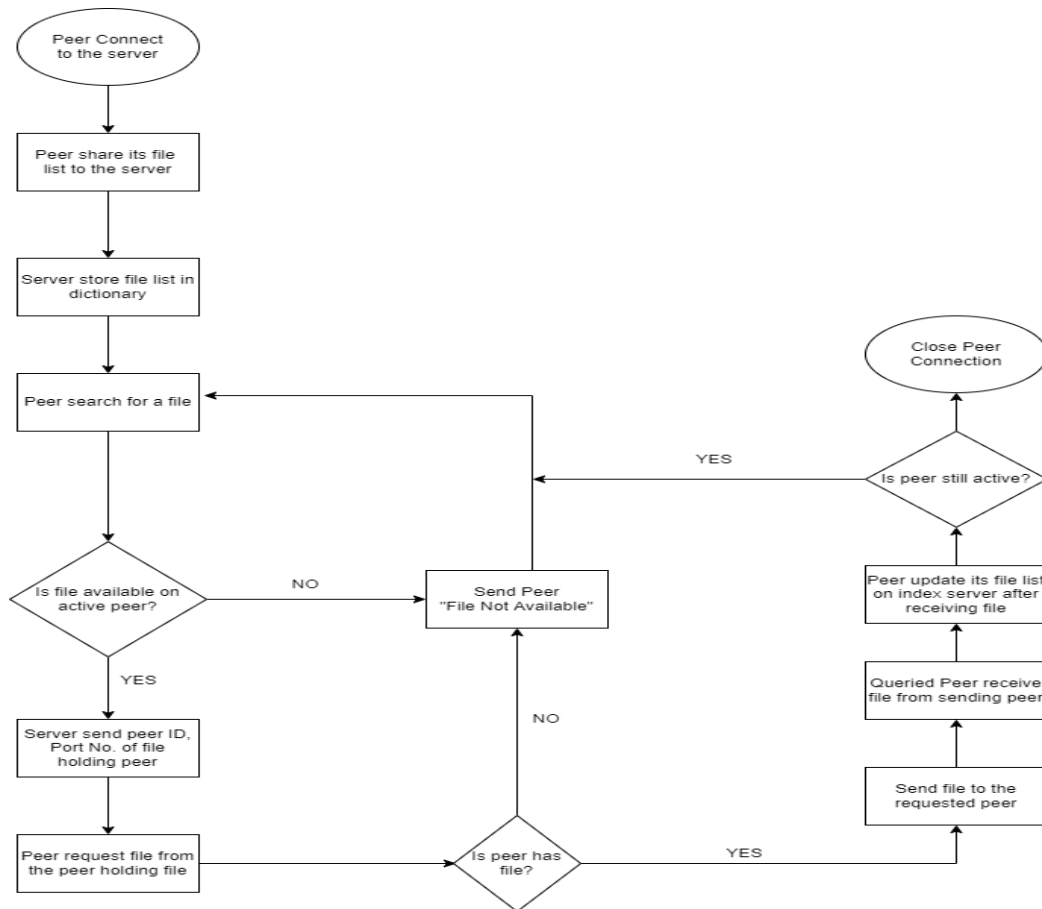
After receiving the request, peer checks the file for availability and sends it to the other peer. Peer is automatically disconnected once file is sent and the thread ends. The file received by the peer gets updated to the indexing server.

The most important part of the project is the use of threads. It is implemented at server side to handle multiple request. At the server (on peer and central indexing server) each client is handled by the separate threads. They are designed such a way that the global variables are read only.

How the complete system design works is depicted in the figure below:

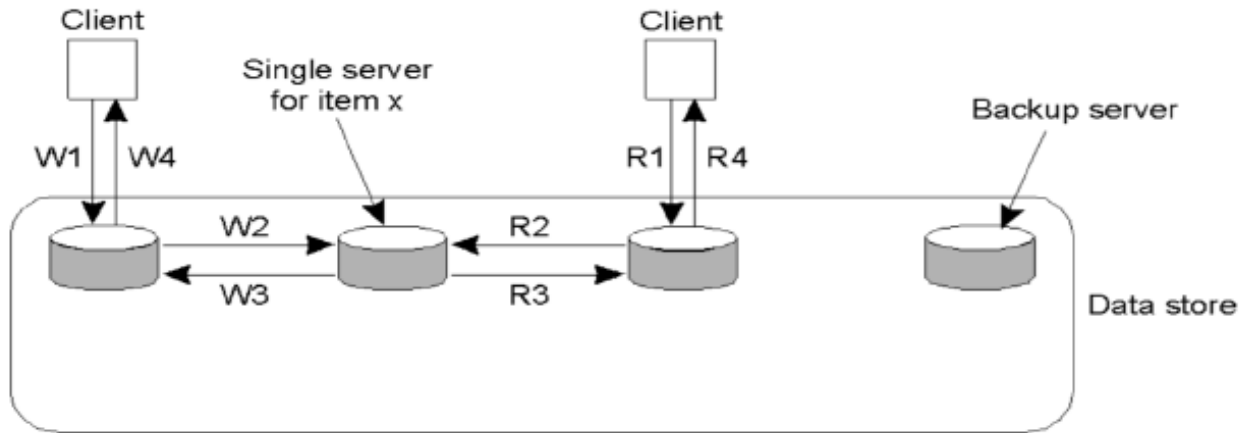


Flow Chart:



### How we can make this system fault tolerance:

The main limitation of this system is the Single Point of Failure. As we know, System design only contains just one index server which is used to store the file indexes of all connected active peers. If the Index Server crashes or is struck into the error there will be the complete system communication cut off. There are two approaches to avoid this failure. One is to add a backup server that continuously checks the status of the index server through heartbeat messages. If the index server goes down then the backup server will get activated and start performing the same responsibility again.



We can also start another server where we have implemented the exception handling in code. Like when server crashed it will stop responding to requests then automatically new server will start but every node needs to connect again and share their files again to the server.

The second approach to avoid a single point of failure is to apply physical redundancy which ensures data security and smooth communication even if one server goes down the other will take the responsibility and in that way the communication goes on. But it will be an expensive approach as we have to add more hardware.

### Snapshots:

We have one server and 3 active peers for project demonstration however we can add more peers through params.py

By using Command line prompt and entering the command python server.py the server has started.

**>> Server is up.**

```
C:\Users\Syed Aitezaz>cd C:\Users\Syed Aitezaz\Desktop\Code
C:\Users\Syed Aitezaz\Desktop\Code>python server.py
->Server up.
```

**>>3 active peers started and get connected to the server.**

```
Command Prompt - python server.py
>File: song.mp3
>File: song1.mpeg
***New client connected with peer ID = 2
>>Client 2 is adding file to its file list
>File: file10_3.txt
>File: file1_3.txt
>File: file2_3.txt
>File: file3_3.txt
>File: file4_3.txt
>File: file5_3.txt
>File: file6_3.txt
>File: file7_3.txt
>File: file8_3.txt
>File: file9_1.txt
>File: file9_3.txt
>File: song.mp3
>File: song1.mpeg

Command Prompt - python peer_2.py
Microsoft Windows [Version 10.0.19044.1645]
(c) Microsoft Corporation. All rights reserved.

C:\Users\Syed Aitezaz>cd C:\Users\Syed Aitezaz\Desktop\Code

C:\Users\Syed Aitezaz\Desktop\Code>python peer_2.py
->Connected to the server at port 5015
***Connection successfull (peerID= 1). Syncing shared folder file list.
->File list is synced.
->Upload server is now running at port 6021
>Please enter the file needed:

Command Prompt - python peer_1.py
Microsoft Windows [Version 10.0.19044.1645]
(c) Microsoft Corporation. All rights reserved.

C:\Users\Syed Aitezaz>cd C:\Users\Syed Aitezaz\Desktop\Code

C:\Users\Syed Aitezaz\Desktop\Code>python peer_1.py
->Connected to the server at port 5015
***Connection successfull (peerID= 0). Syncing shared folder file list.
->File list is synced.
->Upload server is now running at port 6020
>Please enter the file needed:

Command Prompt - python peer_3.py
Microsoft Windows [Version 10.0.19044.1645]
(c) Microsoft Corporation. All rights reserved.

C:\Users\Syed Aitezaz>cd C:\Users\Syed Aitezaz\Desktop\Code

C:\Users\Syed Aitezaz\Desktop\Code>python peer_3.py
->Connected to the server at port 5015
***Connection successfull (peerID= 2). Syncing shared folder file list.
->File list is synced.
->Upload server is now running at port 6022
>Please enter the file needed:
```

### >>All active peers have shared the list of files to index server

Now the index server stores the list of files using data structures dictionaries. It has stored the data in key value pairs. Peer id stored as key and file list is stored as values against the corresponding peer ID.

```

C:\Users\Syed Aitezaz>cd C:\Users\Syed Aitezaz\Desktop\Code
C:\Users\Syed Aitezaz\Desktop\Code>python server.py
->Server up.
***New client connected with peer ID = 0
>>Client 0 is adding file to its file list
>File: 1file10_2.txt
>File: 1song.mp3
>File: file10_1.txt
>File: file10_2.txt
>File: file1_1.txt
>File: file2_1.txt
>File: file3_1.txt
>File: file4_1.txt
>File: file5_1.txt
>File: file6_1.txt
>File: file7_1.txt
>File: file8_1.txt
>File: file8_3.txt
>File: file9_1.txt
>File: song.mp3
>File: song1.mpeg
***New client connected with peer ID = 1
>>Client 1 is adding file to its file list
>File: file10_2.txt
>File: file1_2.txt
>File: file2_2.txt
>File: file3_2.txt
>File: file4_2.txt
>File: file5_2.txt
>File: file6_2.txt
>File: file7_2.txt
>File: file8_2.txt
>File: file8_3.txt
>File: file9_2.txt
>File: song.mp3
>File: song1.mpeg

```

**>>To show the exchange of files Peer 0 ask the index server for file “file9\_2.txt”**

**>>Server Window:**

```

>>Client 0 queried file named file9_2.txt.
>>Queried file file9_2.txt by Client 0 is available.
> @ 1

```

**>> Peer 0 Window:**

It shows the server responded that only active peer 1 has the queried file. Now peer 0 will confirm that he wants the queried file from peer 1 and will establish P2P communication between peer 0 and peer 1.



```

C:\Users\Syed Aitezaz\Desktop\Code>python peer_1.py
->Connected to the server at port 5015
***Connection successfull (peerID= 0). Syncing shared folder file list.
->File list is synced.
->Upload server is now running at port 6020
>Please enter the file needed: file9_2.txt
*File found in active peers.
> file@ peerID: 1
>Enter peerID to select peer (Enter cancel to cancel):

```

**>> Peer 0 Window:**

**>>Peer 0 successfully download the queried file from Peer 1 through P2P Communication.**

```

C:\Users\Syed Aitezaz\Desktop\Code>python peer_1.py
->Connected to the server at port 5015
***Connection successfull (peerID= 0). Syncing shared folder file list.
->File list is synced.
->Upload server is now running at port 6020
>Please enter the file needed: file9_2.txt
*File found in active peers.
> file@ peerID: 1
>Enter peerID to select peer (Enter cancel to cancel): 1
>>Downloading file file9_2.txt.
->Downloaded. Updating indexing server
->Updated

```



















**>>Peer 1 Window:**

```

C:\Users\Syed Aitezaz\Desktop\Code>python peer_2.py
->Connected to the server at port 5015
***Connection successfull (peerID= 1). Syncing shared folder file list.
->File list is synced.
->Upload server is now running at port 6021
>Please enter the file needed:
>>File 0 requested by client file9_2.txt.
>>File file9_2.txt requested by client 0 is available and preparing to send.
>>File file9_2.txt requested by client 0 was sent.
->Disconnecting client with ID 0
>>Please enter the file needed:

```

>>The requested file is now present in the file folder of Peer 0:

 .DS_Store	04/07/2018 7:05 AM	DS_STORE File	7 KB
 1file10_2.txt	24/05/2022 1:27 AM	Text Document	10 KB
 1song.mp3	24/05/2022 12:43 AM	MP3 File	3,491 KB
 file1_1.txt	04/07/2018 7:05 AM	Text Document	1 KB
 file2_1.txt	04/07/2018 7:05 AM	Text Document	2 KB
 file3_1.txt	04/07/2018 7:05 AM	Text Document	3 KB
 file4_1.txt	04/07/2018 7:05 AM	Text Document	4 KB
 file5_1.txt	04/07/2018 7:05 AM	Text Document	5 KB
 file6_1.txt	04/07/2018 7:05 AM	Text Document	6 KB
 file7_1.txt	04/07/2018 7:05 AM	Text Document	7 KB
 file8_1.txt	04/07/2018 7:05 AM	Text Document	8 KB
 file8_3.txt	04/07/2018 7:05 AM	Text Document	8 KB
 file9_1.txt	04/07/2018 7:05 AM	Text Document	9 KB
 file9_2.txt	26/05/2022 12:05 AM	Text Document	9 KB
 file10_1.txt	04/07/2018 7:05 AM	Text Document	10 KB
 file10_2.txt	24/05/2022 1:10 AM	Text Document	10 KB
 song.mp3	24/05/2022 12:41 AM	MP3 File	3,491 KB
 song1.mpeg	21/05/2022 11:26 PM	MPEG File	5,939 KB