Mini-Napster Project (P2P Music Sharing)

Gabe Sanders
gabe.sanders@plu.edu

Max Lopez-Garibaldi
lopezgm@plu.edu

Seven Son
sson@plu.edu

***Abstract*—Napster, launched in 1999 by Shawn Fanning and Sean Parker, was a peer-to-peer file-sharing service that allowed users to easily search for and download MP3 music files from others through a centralized database, gaining massive popularity especially among college students. This project aims to recreate that original 1999 Napster architecture, which used a centralized server to catalog songs shared by users and connect those seeking to download the same tracks [1][2].**

Keywords—napster, udp, p2p, file sharing, networking

I. Introduction

Napster was a peer-to-peer (P2P) file-sharing service launched in June 1999 by Shawn Fanning and Sean Parker. It allowed users to easily search for and download MP3 music files from each other's computers through a centralized database, making music sharing incredibly simple and popular, especially among college students. At its peak, Napster had about 80 million registered users and triggered a huge shift in public expectations for access to digital music [1][2].

This project aims to replicate the original 1999 Napster architecture, which included a centralized server that stored and organized a catalog of songs shared by users. The server's role was to connect users looking to download a specific song with those who had that song available for sharing, facilitating efficient peer-to-peer file exchanges.

II. Background

A. Overview of Napster

Napster, launched in June 1999 by college student Shawn Fanning and Sean Parker, was a pioneering peer-to-peer (P2P) file-sharing software that revolutionized how people accessed music online. Unlike previous methods, Napster allowed anyone with the software to search through and download MP3 music files from other users' computers, making it extremely easy to find and share songs for free. The platform rapidly grew in popularity—especially among students—leading to tens of millions of users by the year 2000 [1][2].

Napster's approach bypassed traditional music distribution, so fans could access a vast library of tracks without purchasing albums or singles. This massive, free exchange of copyrighted music led to swift backlash from the music industry, with the Recording Industry Association of America (RIAA) and famous artists like Metallica filing lawsuits by late 1999. Although Napster eventually faced court-ordered shutdown, its impact was profound, forever transforming expectations around digital music and paving the way for future streaming services [1].

The goal of this project is to recreate the original architecture of Napster in 1999, using a centralized server to store and categorize different songs users want to share, and connecting users to other users that want to download that same song. The design is supposed to demonstrate how peer-to-peer technology works and how it can be leveraged to create applications with unlimited scale.

B. Original Napster System Design

Napster's original design combined a centralized server with peer-to-peer (P2P) file sharing. The central server maintained an index catalog of which users had which MP3 files, storing metadata like filename, IP address, and port number. When a user searched for a song, the server returned a list of peers that had the file. File transfers then happened directly between users without involving the central server further. This hybrid approach gave Napster efficient search and discovery while leveraging P2P for file downloads [3].

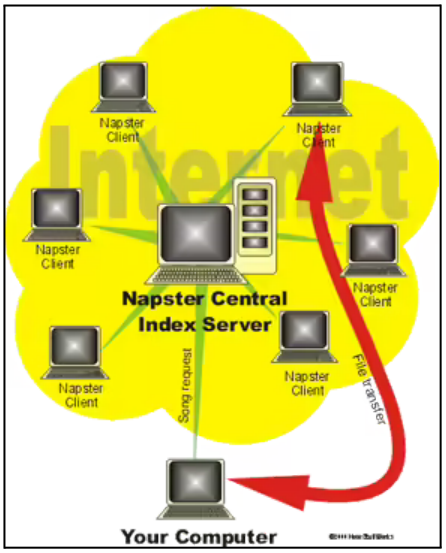

Fig 1. Example of how Napster file sharing worked [3]

For actual file sharing, Napster used the UDP (User Datagram Protocol) to enable fast, direct data transfers between user machines. UDP allowed streaming of music files with lower overhead and latency compared to TCP, though without guaranteed delivery. Transfers occurred directly between peers' computers over the internet, making the sharing decentralized in terms of file exchange but still reliant on centralized indexing to connect users. This design enabled fast, large-scale music sharing while making Napster vulnerable to legal challenges targeting its central server function [3].

C. Xnap - Spiritual Successor to Napster

XNap is a plugin-enabled framework for peer-to-peer (P2P) applications, accompanied by a client based on this framework. Written entirely in Java, XNap is free software under the GNU Public License and features a modern Swing-based user interface that runs on any platform with Java Runtime Environment 1.3 or higher [4].

The project's mission is to help developers focus on core P2P technology by providing easy-to-use APIs for common

tasks like search, transfer, and collaboration, while allowing protocol-specific extensions through a common interface. The client is designed to be largely dependent on plugins, which provide various network protocol implementations, including front ends for existing protocols such as Gnutella [4].

Essentially, this project allows people to both self-host their own central server to do P2P connections between users and provides a customizable client to allow users to search for files that are beyond the MP3 files Napster was designed for.

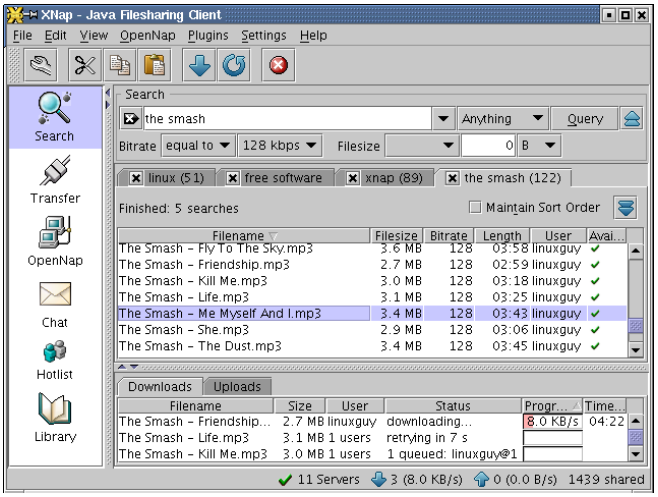

Fig 2. Xnap Client Screenshot [4]

III. Our System Design

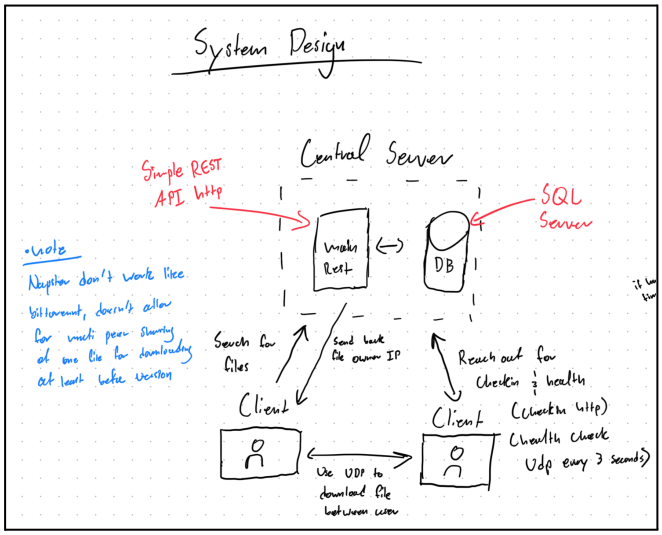

Fig 3. Overall System Design

Central Server - The central server is used to keep track of users who are online and keep track of which files are active to be shared across users. The central server is also in charge of matching user with other users in order to download files. This is all done via a REST API for search for and allowing users to check-in.

For client health checking this will be done via UDP connection, since file transfers happen via UDP and UDP is much quicker for health checks for each client. Health checks should be done every 3 seconds. Clients that failed the health check will be marked as dead in the database. Since this is a toy project, there will be no authentication system put in place in order to streamline everything and increase development productivity.

Database - The database is used to keep track of users and different songs that users want to share.

Client - The client allows users to check in into the central server and search for different music files they want to download. Once a user has selected a music file to download then they will be given the other client's IP address and port, as well as other metadata to then connect to the other user and start downloading the file.

For security purposes the client will verify if a file is correct by comparing the SHA256 (checksum) values to see if the file has been downloaded correctly (this is just simple since this is a toy project and not going to be in production). Similar to how the original Napster worked, only MP3 files will be allowed to be shared, since they are very compressed and small in size.

A. *Database Layout Central Tracker*

Within our database we have three main tables to keep track of everything within our central tracker. All tracking is done in memory and uses tables to manage everything.

Songs - This is a table to keep track of the different songs being tracked. Song will have metadata that can be used for searching, such as by song name, album, and artist; this will be taken from the MP3 metadata. The primary key is FileId, which is a UUID. FileId is used to link to both the Hashes table and Peers table.

Hashes - This table is used to store the SHA256 (checksum) value of each music file being tracked. This is used to verify if a download was correct by a client.

Peers - This table is used to keep track of which files a user is sharing. Peers also keep track of which users are active and can share a specific music file. This table also stores the user IP and port values for UDP connections.

B. *REST API for Central Tracker*

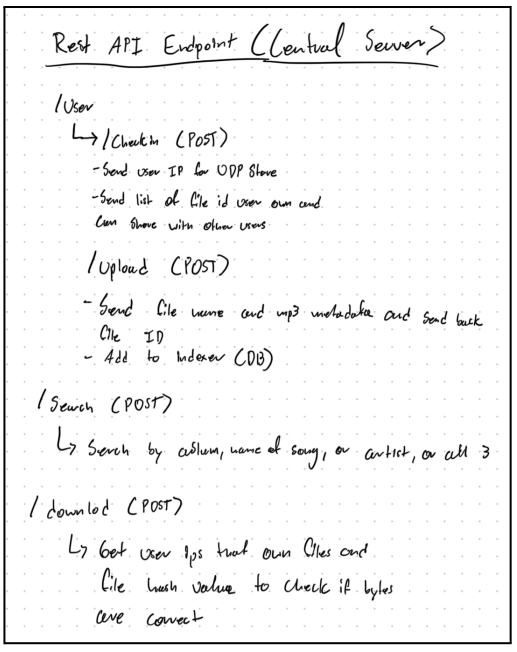

Fig 4. REST API for Central Tracker Server

Fig.4 represents the different REST API endpoints that a client can interact with in order to search, upload, and download music files. The reason for everything to be a POST request is to make client implementation easier, since we can make each request body an object and have it be much more flexible for development.

User - The User endpoint is designed to allow a user to check in with the central server as well as ask to have a file to be tracked, so other users can download it. As stated before, given that the scope of this project is to be a demo of how Napster works, no authentication will be added to verify users.

Search - The Search endpoint allows a user to search through the database for songs they would like to download.

Download - This endpoint gives the client both the SHA256 (checksum) of the file and the UDP connection information of the other client that has the file in order to start downloading that file.

C. Client Command Line Interface

We use a command-line interface (CLI) that enables users to interact with the peer-to-peer file-sharing network. Upon initialization, users specify their username, IP address, and port number as command-line arguments, which configure their network identity and listening parameters. The interface displays a welcome message and initializes a NapsterServer instance that manages the user's connection to the network and handles incoming requests from other peers. The CLI operates in a continuous loop, accepting user commands and routing them to appropriate command handlers stored in the commands module, while maintaining a persistent pool of download clients to optimize transfer operations across multiple file requests.

The system supports six primary commands that facilitate core peer-to-peer operations:

- **search-file (aliases: sf)** allows users to query the network for files by artist and song name;
- **leachers (aliases: lr)** displays information about peers downloading from the current user;
- **shared-list (aliases: sl)** lists all files the user is currently sharing;
- **download (aliases: dl)** initiates file transfers from discovered peers, with client connections cached for efficiency;
- **downloads (aliases: dls)** shows the status of ongoing or completed downloads; and
- **clear (aliases: cls**) clears the terminal display. Additionally, users may enter the exit command to terminate the application gracefully.

This modular command architecture allows for extensibility, enabling researchers to add new functionality by implementing additional command modules without modifying the core CLI logic.

IV. SHOWCASE

Our current implementation is done using Python and the socket library offered as part of Python standard library [6]. We also use SQLite to manage the client side database and files being shared between clients [5].

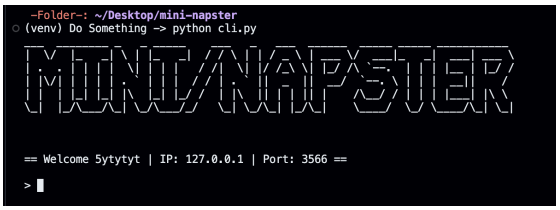

Fig 5. Screenshot of Welcome CLI

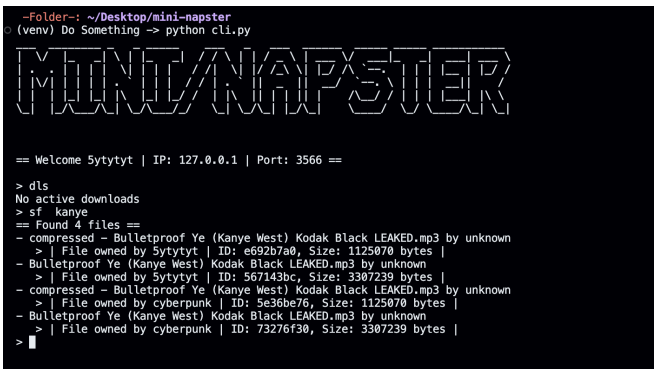

Fig 6. Screenshot of search command

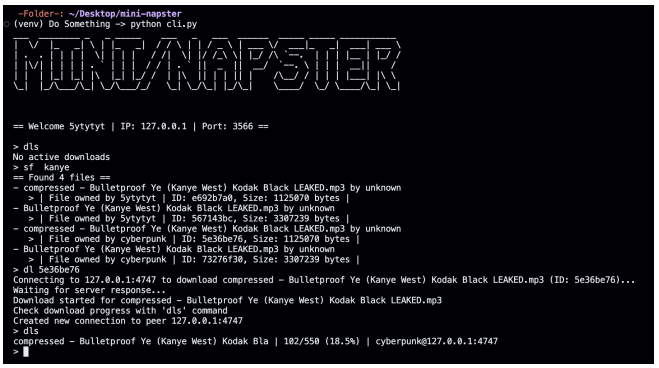

Fig 7. Screenshot of downloading a song

Fig 8. Screenshot of who downloading files

All the screenshots above showcase the general functionality of the mini-napster project and how a user would search for and download music of their choice.

V. CONCLUSION

Developing "Mini-Napster" provided critical insights into the intricacies of peer-to-peer architectures and the trade-offs inherent in network protocols. We observed that while UDP offers significant performance benefits for file transfers due to its low overhead, it necessitates robust application-level mechanisms to handle packet loss and ordering—complexities that TCP typically manages abstractly. Furthermore, implementing a hybrid architecture, which utilizes a central tracker for metadata while offloading actual data transfer to peers, highlighted the efficiency of decentralized networks in reducing server bottlenecks. This experience reinforced the importance of

precise protocol design and the challenges of maintaining consistent state across a distributed system.

However, the development process presented significant hurdles, primarily driven by strict time constraints that forced several strategic pivots. To ensure a reliable demonstration within the deadline, we migrated the central server from a persistent MySQL database to an in-memory implementation, prioritizing immediate access speed over long-term persistence. We also navigated architectural complexities within Python, specifically resolving circular import cycles caused by interdependent class structures. Finally, while a graphical interface was initially planned, the complexity of integrating Python GUI frameworks proved prohibitive given the timeframe. Consequently, we shifted to a streamlined Command Line Interface (CLI), a decision that not only accelerated development but also endowed the application with a "hacker-style" terminal aesthetic that complemented its technical nature.

References

[1] Michael, P. (2023). Napster is released: Research starters: EBSCO research. EBSCO. https://www.ebsco.com/research-starters/computer-science/napster-released

[2] Software, B. (2023). 1999 in tech: Napster and the rise of peer-to-peer file sharing. 1999 in Tech: Napster and the rise of Peer-to-Peer File Sharing. https://blog.boson.com/1999-in-tech-napster-and-the-rise-of-peer-to-peer-file-sharing

[3] Tyson, J. (2000, October 30). How the old napster worked. HowStuffWorks. https://computer.howstuffworks.com/napster.htm

[4] Overview. XNap. (n.d.). https://xnap.sourceforge.net/about.html

[5] SQLite Home Page. (2019). Sqlite.org. https://sqlite.org/

[6] PYTHON. (2025). Python. Python.org; Python.org. https://www.python.org/