|  |
| --- |
|  |
| Fuzzy Hipster Audio Player |
| Version 1.6 |
|  |
| **Aaron Lee (A00621294)**  **Behnam Bastami (A00197752)**  **Ronald Bellido (A00694189)**  **Jesse Braham(A00668867)** |
| **18/03/2013**  **Revised: 24/3/2013 (v1.4)**  **Revised:2/4/2013 (v1.5) [new protocols]**  **Revised:2/4/2013 (v1.6) [testing / user guide, new request]** |

Table of Contents

[Comm Audio Player Design 4](#_Toc352703900)

[Requests 4](#_Toc352703901)

[Upload Request type 4](#_Toc352703902)

[Download Request type 4](#_Toc352703903)

[Stream Request type 5](#_Toc352703904)

[2-Way Microphone Request type 5](#_Toc352703905)

[Technologies/Libraries used 5](#_Toc352703906)

[Client 6](#_Toc352703907)

[Overview 6](#_Toc352703908)

[Overall STD 6](#_Toc352703909)

[Overall Pseudocode 7](#_Toc352703910)

[File Download/Upload 8](#_Toc352703911)

[Download/Upload STD 8](#_Toc352703912)

[Download Pseudocode 9](#_Toc352703913)

[Upload Pseudocode 9](#_Toc352703914)

[Streaming 10](#_Toc352703915)

[Streaming STD 10](#_Toc352703916)

[Streaming Pseudocode 10](#_Toc352703917)

[2-way Microphone Chat 11](#_Toc352703918)

[2-way microphone STD 11](#_Toc352703919)

[2-way microphone Pseudocode 11](#_Toc352703920)

[Multicasting 12](#_Toc352703921)

[Multicast STD 12](#_Toc352703922)

[Multicast Pseudocode 12](#_Toc352703923)

[UI 13](#_Toc352703924)

[Server 14](#_Toc352703925)

[State Transition Diagrams 14](#_Toc352703926)

[Overview 14](#_Toc352703927)

[Overall Server Pseudocode 15](#_Toc352703928)

[TCPListen: 15](#_Toc352703929)

[ListenForClientRequests: 15](#_Toc352703930)

[Handle Upload Request 17](#_Toc352703931)

[Handle Upload Requests Pseudocode 18](#_Toc352703932)

[Handle Download Request 19](#_Toc352703933)

[Handle Stream Request 20](#_Toc352703934)

[Handle 2-way Microphone Chat Request 21](#_Toc352703935)

[UI 22](#_Toc352703936)

[Testing and Verification 23](#_Toc352703937)

[User Manual 25](#_Toc352703938)

[Server 25](#_Toc352703939)

[Client 26](#_Toc352703940)

# Comm Audio Player Design

Comm Audio Player is a streaming audio player, capable of playing audio streams from a network server, establishing a 2-way microphone chat, downloading and uploading a song on the server and listening to a multicast channel.

Comm Audio Player consists of a server and a client, and at startup, the user can choose to launch Comm Audio Player as one or the other. As a client, the user can specify whether they want to stream a song, upload or download a song from the server, start a 2-way microphone chat, or listen to a multicast channel. The server, assuming it is established before, is able to listen for new clients, while handling requests from multiple clients.

## Requests

Each request is a packet that is sent by the client to the server. A request packet will always consist of the following:

[header | (optional: data) | \n]

* The format of this packet shall be in string format, and is terminated by new line character.
* *header* will be dependent on what type of the request the packet will be.
* *data* is dependent on the type of request and may be optional on some requests.
* A sample of a typical request would be: “UL Behnam’s Party Mix.wav\n”
  + This is an upload request for the song ‘Behnam’s Party Mix.wav’

### Upload Request type

[ header | filename | \n]

*Header type: “UL”*

*Filename* is a string that will be used by the server to name the file being uploaded

**Note**: the server will reply, if the upload request is granted, with the exact same packet. If the request is rejected, the server will reply with the same packet but the *filename* will be an empty string. Reasons for an upload request to be rejected may include: client is currently downloading a file from the server, song list is not yet current.

### Download Request type

[ header | filename | \n]

*Header type: “DL”*

*Filename* is a string name of a song that must exist on the server.

**Note**: the server will reply, if the download request is granted, with the exact same packet. If the request is rejected, the server will reply with the same packet but the *filename* will be an empty string. Reasons for an download request to be rejected may include: client is currently downloading a file from the server, song list is not yet current.

### Stream Request type

[ header | filename index | \n]

*Header type: “ST”*

*Filename Index:* The index of the song requested according to the current song list.

**Note**: the server will reply, if the streaming request is granted, with the exact same packet. If the request is rejected, the server will reply with the same packet but the *filename* will be an empty string. Reasons for a streaming request to be rejected may include: the file requested does not exist in the server.

### 2-Way Microphone Request type

[ header | \n]

*Header type: “MIC”*

### Multicast type

[ header | \n]

*Header type: “MC”*

### List type

[ header | \n]

*Header type: “LIST”*

**Note**: the server will reply with a new-line separated string of music files currently available on the server

## Technologies/Libraries used

* libZPlay Win32 Audio Library
  + Used for audio components of the application
* Windows Sockets API
  + Used for the networking component of this application

# Client

## Overview

### Overall STD



### Overall Pseudocode

#### Get User Settings:

* Allow user to choose between server and client modes
* Allow user to enter server address and port number
* Validate settings; if invalid, display error and prompt user to input valid values

#### Create control channel

* Create a TCP socket
* Connect the socket to the servers TCP socket
* Retrieve song listing from server and populate song list on GUI
* Retrieve connection info (port number) for multicast from server

#### Wait for user command

* This is the listen for command state on client
* Valid user commands are :
  + - * Request to Download a music
      * Request to Upload a music
      * Request to stream a music
      * Request to start 2-way microphone communication
      * Join the server’s multicast channel

## File Download/Upload

### Download/Upload STD



### Download Pseudocode

#### Send DL request

* Create a new thread
* Send a packet to the server requesting a file download
* The request is in the following form
  + “DL FILENAME\n”

#### Wait for file

* In the download thread wait for the file packets to arrive

#### Save to file

* While receiving file packets, save to a file
* When number of bytes equal to file size is read, stop downloading and return to user command state

### Upload Pseudocode

#### Send UL request

* Create a new thread
* Send a packet to the server requesting a file upload
* The request is in the following form
  + “UL FILENAME\n”

#### Wait for approval

* In the upload thread wait for server response
* If approved, server will echo the request packet back to the client

#### Send File

* While not EOF Packetize the file and send over TCP
* When finished, go back to listen for command mode

## Streaming

### Streaming STD



### Streaming Pseudocode

#### Send stream request

* Send a packet to the server requesting a file stream
* The request is in the following form
  + “ST FILEINDEX”
    - Filename index is the index of the song requested according to the current song list

#### Stream song

* Create a new thread and wait for the file packets to arrive
* While receiving file packets, play the music
* If a stream is already in progress when another stream request is made
  + Kill the previous stream
* Send a new stream request

## 2-way Microphone Chat

### 2-way microphone STD



### 2-way microphone Pseudocode

* Stop any other streaming that is in progress
* Create a microphone UDP channel on a new thread
* When the user starts the microphone chat, read data from microphone and send to server over the UDP channel
* When the user stops the microphone chat, perform cleanup and close the socket

## Multicasting

### Multicast STD



### Multicast Pseudocode

* Stop any other streaming that is in progress
* Create a new thread to join the server’s multicast channel
* While receiving packets on the multicast channel, play the music data

### UI



# Server

## State Transition Diagrams

### Overview



### Overall Server Pseudocode

#### StartMulticast:

* Get current song list
* Broadcast first song on the list
* Continue to broadcast, looping the song list over and over
* If application is terminated:
  + Call BroadcastTeardown()
  + Exit

#### BroadcastTeardown:

* Stop playing song
* Close broadcast socket

#### BuildSongList:

* Scan “music” folder
* Create vector of song list

### TCPListen:

* Listen for new client connections
* If a new client connects:
  + Create a new thread:
    - Send current song list to client
    - Call ListenForClientRequests()

### ListenForClientRequests:

* While (true):
  + If request is received:
    - Call DecodeRequest()

#### DecodeRequest:

* Check the header type of received packet
  + If “UL”:
    - if not currently in *download state*:
      * Start ReceiveMode
    - Else
      * Send client with the packet: [header | “ ” | \n]
  + If “DL”:
    - Send the song located at *filename index* in the song list vector
  + If “ST”:
    - If current state is *streaming*:
      * End current stream
    - Start streaming song located at *filename index* in the song list vector
  + If “MIC”:
    - If current state is *streaming*:
      * End the current stream
    - Create thread:
      * Keep streaming microphone
      * If microphone stream ended:
        + If socket is not closed:

Display error

* + If “MC”
    - Create thread:
      * For each file in music directory, open then write to UDP socket
  + If “LIST”
    - Read file names of all files in music directory
    - Concatenate all file names into a string message
    - Send string message to client

### Handle Upload Request



### Handle Upload Requests Pseudocode

#### Receive Mode

* Receive the file until done
* Append new song into song list
* Send new song list to every client connected

**Note:** Pseudocode for the rest of handling upload requests are in DecodeRequest pseudocode stated in the Server Overview. States are drawn here for clarity.

### Handle Download Request



**Note:** Pseudocode for handling download requests are in DecodeRequest pseudocode stated in the Server Overview. States are drawn here for clarity.

### Handle Stream Request



**Note:** Pseudocode for handling stream requests are in DecodeRequest pseudocode stated in the Server Overview. States are drawn here for clarity.

### Handle 2-way Microphone Chat Request



**Note:** Pseudocode for handling 2-way microphone requests are in DecodeRequest pseudocode stated in the Server Overview. States are drawn here for clarity.

### Handle List Requests Pseudocode

#### Receive Mode

* Read file names of all files in music directory
* Concatenate all file names into a string message
* Send string message to client

# Testing and Verification

Our application was tested against the following conditions:

|  |  |  |
| --- | --- | --- |
| **Condition** | **Verification** | **Screenshot** |
| Files are uploaded, intact, correct name | File transfers from client to server, file is present in appropriate directory with proper name. File plays without issues. |  |
| File list on server is updated accordingly | Upon connecting to the remote server or uploading a file, the list is updated. |  |
| Files are playable from the server | Played a 2.5 MB PCM WAV file and a 10 MB 44kHz 256 kbps CBR MP3 file successfully. | Not Applicable |
| Files are downloaded, intact, correct name | File transfers from server to client, file is present in appropriate directory with proper name. |  |
| Downloaded files are playable on the client (not necessarily from our application) | Upon completion of file download, the audio file is playable in any media player. |  |
| Streaming music is audible with no distortion on the client | Upon a stream being initiated, the music can be heard clearly. | Not Applicable |
| Client can drop in and listen to music with no distortion (multicast) | Client joins multicast session and hears the music clearly. | Not Applicable |
| Client/Server communicate with no distortion, no latency (microphone) | Voices can be heard on both ends with little to no latency or distortion. | Not Applicable |
| Client exits cleanly | A client, once joined, terminates at any point throughout execution without errors. |  |

# User Manual

*System requirements:* Windows XP SP2 or later, libzplay.dll binary, FuzzyPlayer server (fuzzyserver.exe) and client (fuzzyclient.exe) binaries, TCP/IP connectivity, sound card, speakers and a microphone

## Server

The server component of this application consists of a simple command line application. This application takes no command line arguments, and requires no user input throughout the duration of its execution.

To run this application, simply run the “*FuzzyServer.exe*” executable file, or run it from the command line using “*FuzzyServer.exe*”.

Upon execution, in its initial state, the server will look similarly to as follows:

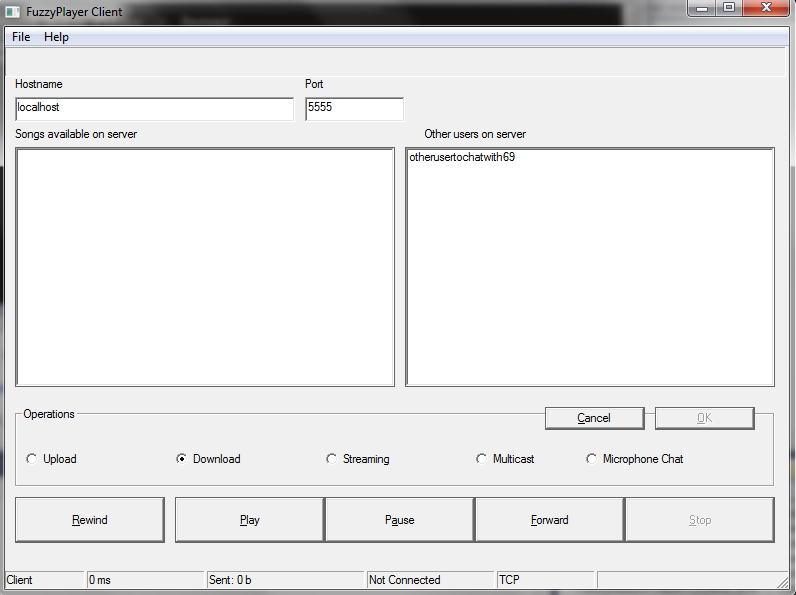


The server will look for audio files in a very particular way. In the same directory that the executable is located, (example, *C:\Users\Admin\Desktop\*), there must be a directory named “*Music*” (example, *C:\Users\Admin\Desktop\Music\*). It should be noted that only supported audio files should be located in this directory.

The server also supports UDP multicasting. This function is automatically started upon execution of the server. It will cast all songs in your “*Music*” subdirectory in order until all have been played. Like a radio station, clients can “tune in” to the FuzzyServer multicast at and time while it is casting.

## Client

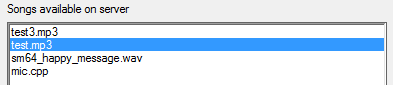
The FuzzyPlayer Client is used to connect to a central server that hosts music files, chat sessions and multicast sessions. The libzplay.dll must be in the same directory as your fuzzyclient.exe executable. Before starting the client, ensure that your microphone and speakers are not muted and plugged in. Ensure the server is running and be sure to know its IP.



To connect to the FuzzyServer:

1. Type in the IP of the computer that is running FuzzyServer into the Hostname box
2. Type in the port of the FuzzyServer (default: 5555)
3. Click on “File”
4. Click on “Connect”

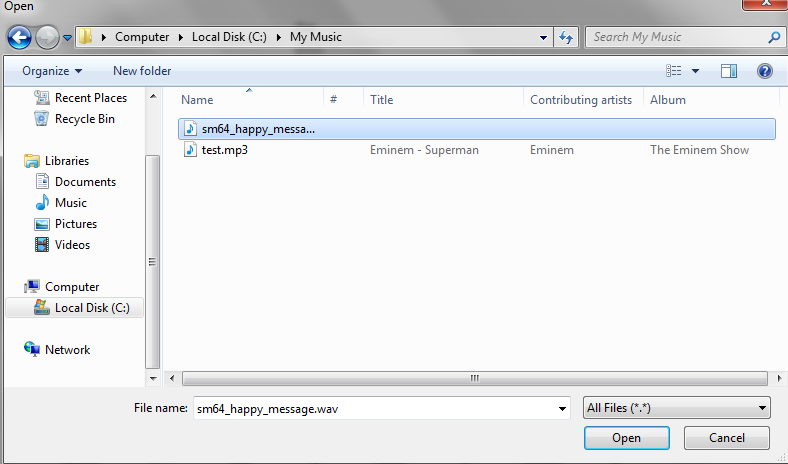
To stream or download music from the server:

1. Connect to the FuzzyServer
2. Click on the song you which to download or stream
3. If you wish to save the file to your local computer, select the “Download” radio button or if you wish play the file, click on “Streaming” radio button
4. Click “OK” and music should start playing or being downloaded to your computer in the same running directory as your FuzzyClient.exe

To start a voice/microphone chat session with the server

1. Ensure that your microphone is enabled, unmuted and plugged into your computer
2. Connect to the FuzzyServer
3. Click on “Microphone Chat” radio button
4. Click OK
5. Click OK on the message box
6. Speak into your microphone to chat with the user on the server side

To upload a music file to the server

1. Connect to the FuzzyServer
2. Click on the “Upload” radio button
3. Click OK
4. On the open file dialog window, select the music file you wish to upload then click “Open”
5. The file will now appear on the server for others to stream and download

To connect to the server multicast session

Note: the server will cast all songs in the song list but, given the nature of multicasting, the user cannot control what the server is now playing.

1. Connect to the FuzzyServer
2. Click on the “Multicast” radio button
3. Click OK