|  |
| --- |
|  |
| Fuzzy Hipster Audio Player |
| Version 1.5 |
|  |
| **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]** |

Table of Contents

[Comm Audio Player Design 4](#_Toc351320086)

[Requests 4](#_Toc351320087)

[Upload Request type 4](#_Toc351320088)

[Download Request type 4](#_Toc351320089)

[Stream Request type 4](#_Toc351320090)

[2-Way Microphone Request type 4](#_Toc351320091)

[Multicast Request type 5](#_Toc351320092)

[Technologies/Libraries used 5](#_Toc351320093)

[Client 6](#_Toc351320094)

[Overview 6](#_Toc351320095)

[Overall STD 6](#_Toc351320096)

[Overall Pseudocode 7](#_Toc351320097)

[File Download/Upload 8](#_Toc351320098)

[Download/Upload STD 8](#_Toc351320099)

[Download Pseudocode 9](#_Toc351320100)

[Upload Pseudocode 9](#_Toc351320101)

[Streaming 10](#_Toc351320102)

[Streaming STD 10](#_Toc351320103)

[Streaming Pseudocode 10](#_Toc351320104)

[2-way Microphone Chat 11](#_Toc351320105)

[2-way microphone STD 11](#_Toc351320106)

[2-way microphone Pseudocode 11](#_Toc351320107)

[Multicasting 12](#_Toc351320108)

[Multicast STD 12](#_Toc351320109)

[Multicast Pseudocode 12](#_Toc351320110)

[UI 13](#_Toc351320111)

[Server 14](#_Toc351320112)

[State Transition Diagrams 14](#_Toc351320113)

[Overview 14](#_Toc351320114)

[Overall Server Pseudocode 15](#_Toc351320115)

[TCPListen: 15](#_Toc351320116)

[ListenForClientRequests: 15](#_Toc351320117)

[Handle Upload Request 16](#_Toc351320118)

[Handle Upload Requests Pseudocode 16](#_Toc351320119)

[Download Request 18](#_Toc351320120)

[Stream Request 19](#_Toc351320121)

[2-way Microphone Chat 20](#_Toc351320122)

[UI 21](#_Toc351320123)

# 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”*

## 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

* Send a packet to the server requesting a file download
* The request is in the following form
  + [ header | filename ]
    - Header includes the following
      * Type: REQDL
      * Size: size of the filename string

#### Wait for file

* Create a new thread and wait for the file packets to arrive

#### Save to file

* While receiving file packets, save to a file
* When EOF received, return to listen for command state

### Upload Pseudocode

#### Send UL request

* Send a packet to the server requesting a file upload
* The request is in the following form
  + [ header | filename ]
    - Header includes the following
      * Type: REQUL
      * Size: size of filename

#### Wait for approval

* Create a new thread to wait for server approval
* 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
  + [ header | filename index ]
    - Header includes the following
      * Type: REQST
      * Size: sizeof(int)
    - 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

### 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.

### UI

