|  |
| --- |
|  |
| Comm Audio Player |
| COMP 4985 Final Assignment |
|  |
| **Aaron Lee (A00621294)**  **Behnam Bastami (A00197752)**  **Ronald Bellido (A00694189)**  **Jesse Braham(A00668867)** |
| **18/03/2013** |

Table of Contents

[Comm Audio Player Design 3](#_Toc351316893)

[Technologies/Libraries used 3](#_Toc351316894)

[How requests work 3](#_Toc351316895)

[Client 4](#_Toc351316896)

[Overview 4](#_Toc351316897)

[Overall STD 4](#_Toc351316898)

[Overall Pseudocode 5](#_Toc351316899)

[File Download/Upload 6](#_Toc351316900)

[Download/Upload STD 6](#_Toc351316901)

[Download Pseudocode 7](#_Toc351316902)

[Upload Pseudocode 7](#_Toc351316903)

[Streaming 8](#_Toc351316904)

[Streaming STD 8](#_Toc351316905)

[Streaming Pseudocode 8](#_Toc351316906)

[2-way Microphone Chat 9](#_Toc351316907)

[2-way microphone STD 9](#_Toc351316908)

[2-way microphone Pseudocode 9](#_Toc351316909)

[Multicasting 10](#_Toc351316910)

[Multicast STD 10](#_Toc351316911)

[Multicast Pseudocode 10](#_Toc351316912)

[UI 11](#_Toc351316913)

[Server 12](#_Toc351316914)

[State Transition Diagrams 12](#_Toc351316915)

[Overview 12](#_Toc351316916)

[Overall Server Pseudocode 13](#_Toc351316917)

[Handle Upload Request 13](#_Toc351316918)

[Handle Upload Requests Pseudocode 14](#_Toc351316919)

[Download Request 14](#_Toc351316920)

[Stream Request 15](#_Toc351316921)

[2-way Microphone Chat 16](#_Toc351316922)

[UI 17](#_Toc351316923)

# 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)]

A *header* is a struct that contains a *size* and a *type*.

*Data* is dependent on the type of request and may be optional on some requests.

### Upload Request type

[ header | filename ]

*Header type: REQUL*

*Header size: filename* length

*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.

### Download Request type

[ header | filename]

*Header type: REQDL*

*Header size: filename* length

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

### Stream Request type

[ header | filename index]

*Header type: REQST*

*Header size: sizeof(int)*

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

### 2-Way Microphone Request type

[ header ]

*Header type: REQMIC*

*Header size: 0*

### Multicast Request type

[ header ]

*Header type: REQMC*

*Header size: 0*

## Technologies/Libraries used

* QT 5.0.1 framework for Windows
  + Used for the front-end user interface
* Simple and Fast Multimedia Library (SFML) 2.0 Release Candidate
  + Used for streaming and playing audio

# 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

#### BuildSongList:

* Scan “music” folder

### Handle Upload Request



### Handle Upload Requests Pseudocode

#### Receive Mode

### Download Request



### Stream Request



### 2-way Microphone Chat



### UI

