2014

BCIT

Ramzi Chennafi
Shane Spoor
Abhishek Bhardwaj

[COMM AUDIO]

A music/voice streaming unicast/multicast client and server developed for Windows.

CONTENTS

PROJECT PROPOSAL

ABSTRACT

The scope of this project will include two crucial capabilities. The first being the **ability to stream music over a multicast IP**, in a way that the clients will be able to join the specified multicast IP and listen to the stream. While the second is the **ability to stream over unicast TCP**. By doing this, each user will receive their own personal music stream and be able to control the stream using their GUI. A secondary capability that will be part of the unicast portion of this project is the **ability to transfer voice data over UDP to multiple or single clients** also connected to the server through the unicast streaming. The server will also have the capability to perform both multicast streaming and unicast streaming simultaneously.

DEVELOPMENT CHOICES

- Developed for Windows using C and Win32

- GUI developed using Qt

- Uses both UDP and TCP, as well as Multicast IP broadcasting

- Implementation of circular buffers for streaming

- Threading on Server side

- Server made without GUI

- Controls for Playing, Stopping, Skipping back and Skipping Forward music on client side

- Display of album art, song info and server playlist on client side

- Use of SQL Lite for the storing of song libraries server side

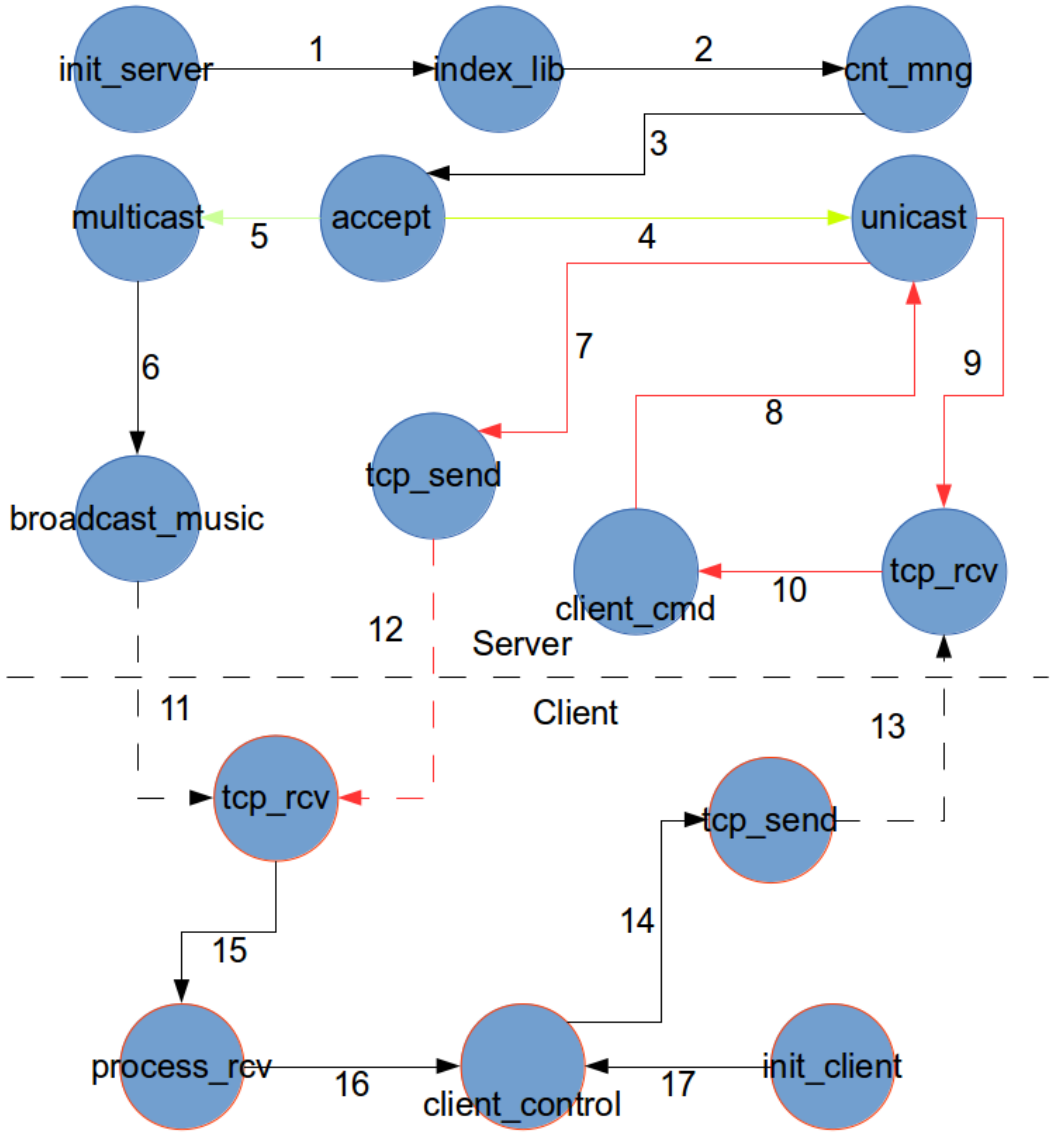
- Possible use of CUnit to perform unit testing (time allowing)

PACKET LISTING

```
typedef struct pkt00{
	char * client_name;
}C_JOIN_PKT;
// Sent on client join of a multi unicast session. Gives the client name
// for addressing purposes for voice.

typedef struct pkt01{
	bool voice_enable;
	bool * client_list;
}C_VOICE_CTRL_PKT;
// Sent whenever the client modifies voice chat options, the client
// may specify if voice is enabled and who they want to listen/talk to.

typedef struct pkt02{
	int control_request;
	char * song_request;
}C_MUSIC_CTRL_PKT;
// Sent whenever a user interacts with the music GUI. Includes definitions:
// Play Music - CLIENT_PLAY
// Pause Music - CLIENT_PAUSE
// Next Song - CLIENT_NEXT
// Previous Song - CLIENT_PREV
// Stop Stream - CLIENT_STOP_UC
// Also allows the user to specify a song to play.

typedef struct pkt03{
	char * audio;
}S_MUSIC_PKT;
// Sent when streaming music to clients.

typedef struct pkt04{
	char * client_name;
	char * audio;
}VOICE_PKT;
// Sent by both the server and client.
// Sent by the client whenever they use their microphone.
// Sent by the server whenever a voice is recieved and a client is
// open for that voice.

typedef struct pkt05{
	char * song_name;
	char * song_length;
	char * artist;
	char * album;
	char * album_art;
}S_SONG_DATA;
// Sends song data whenever a new song starts streaming to the client.
```

PROJECT DIAGRAMS

GENERAL SYSTEM OVERVIEW



A global overlook of the client and server model.

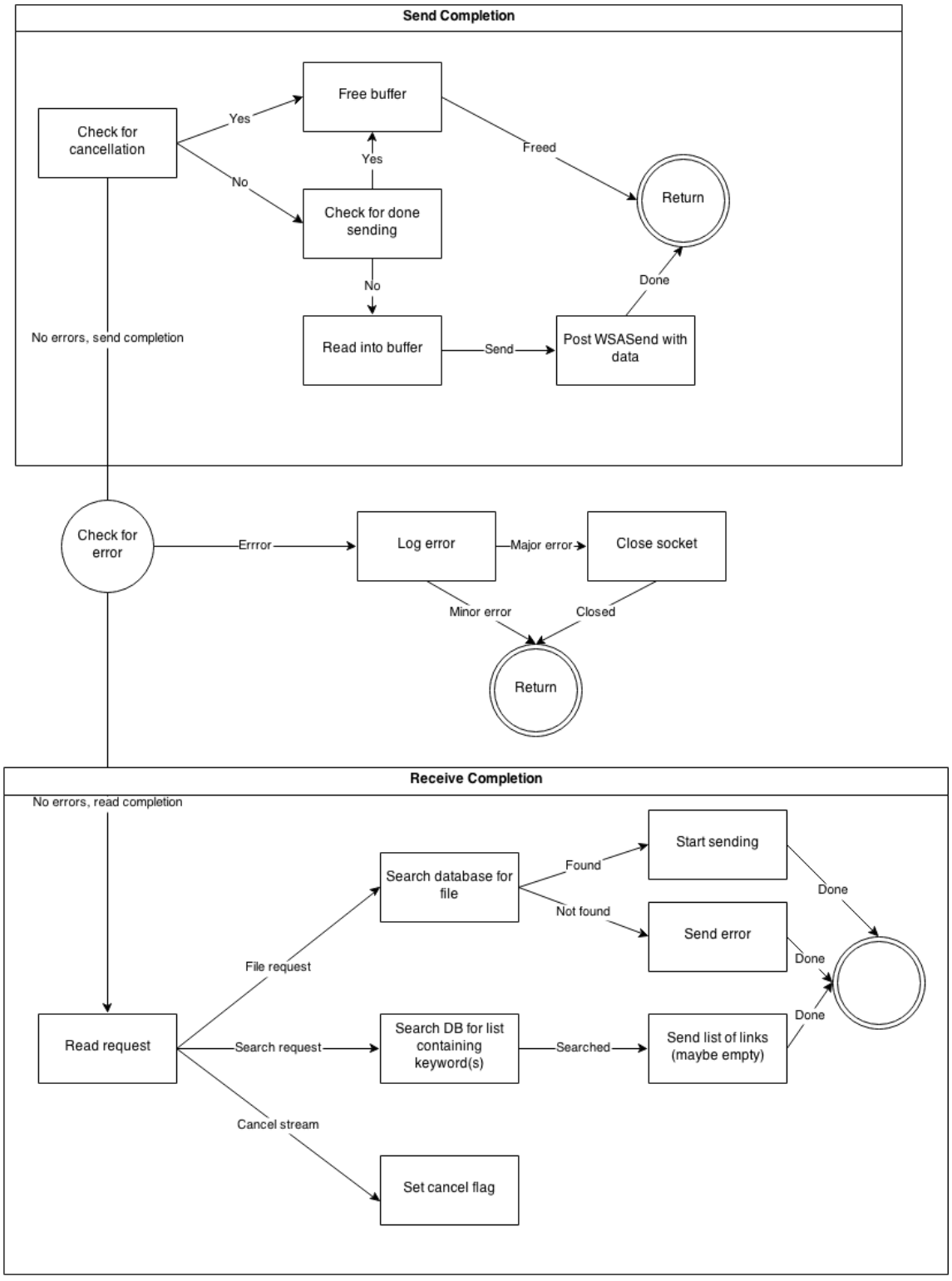
Red lines signify that the operation may be duplicated across several different threads (different client connection unicast threads in this case).

Green lines signify a state where communication occurs between threads.

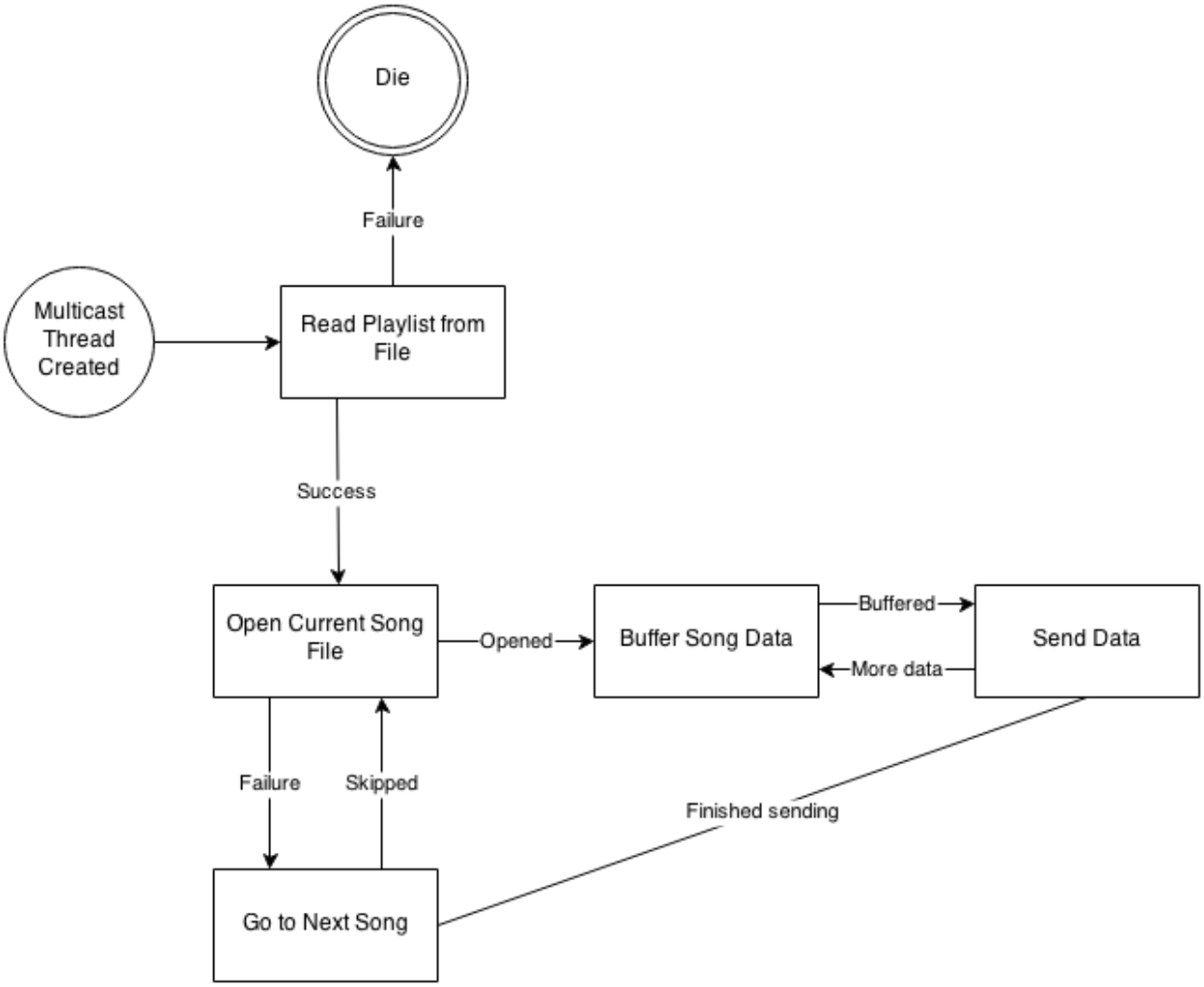
Refer to index below for transition descriptions.

1	Server started up properly.
2	Music library indexing succeeds
3	A new connection request is sent to the server by the client
4	The client requests unicast mode, passes socket descriptor to unicast thread
5	The client requests multicast mode, passes socket descriptor to multicast thread
6	Multicast is able to grab audio stream
7	Server sends data packet, audio or control to client
8	Requested client action is successfully applied
9	Client sends data to the server
10	Data recieved from client is valid command.
11	Multicast data is broadcasted
12	Data is sent from server.
13	Client sends command.
14	User interfaces with client, causes command to be sent
15	Packet is recieved
16	Packet processing is successful
17	Client successfully initiates

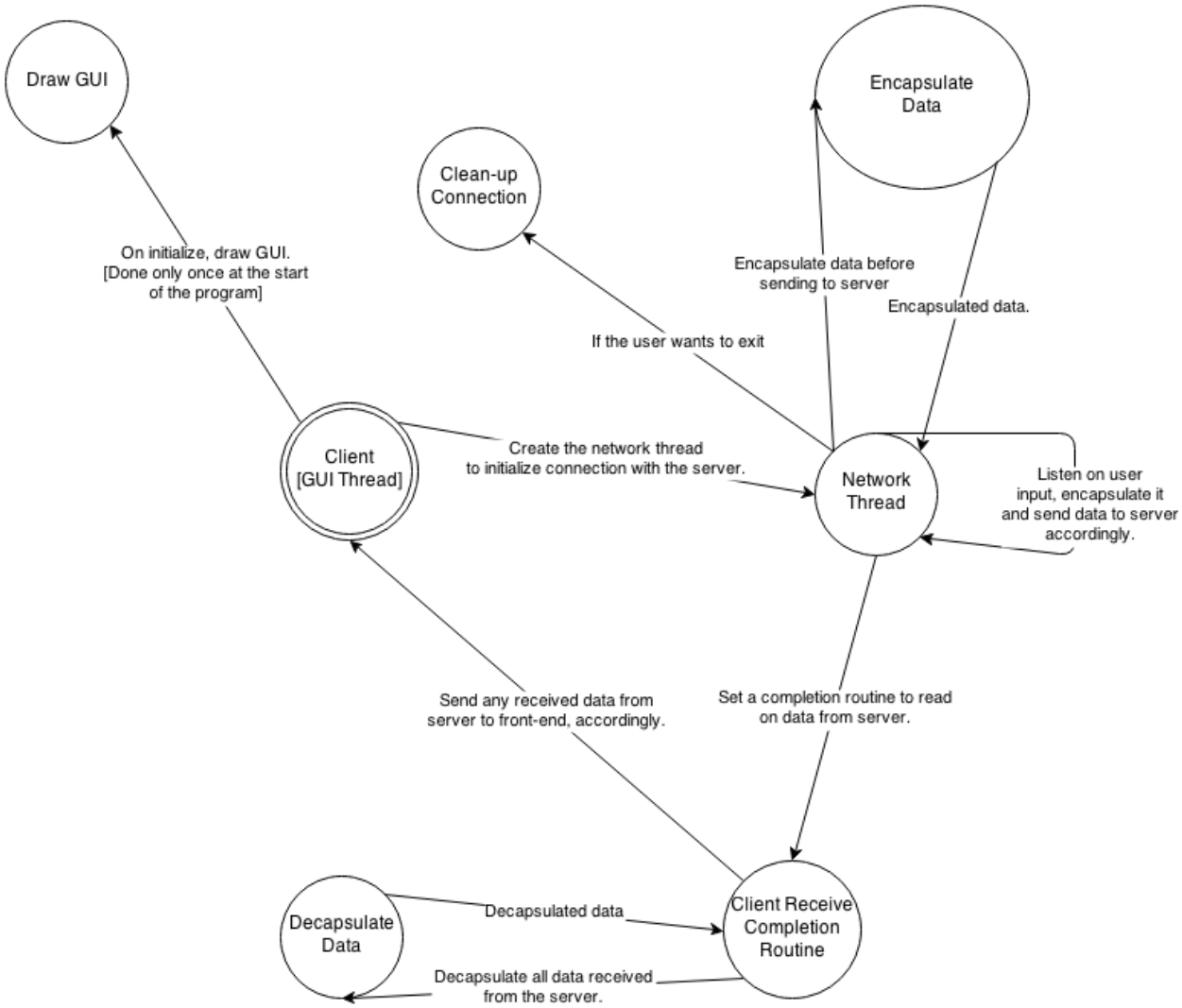
SENDING AND RECEIVING SERVER SIDE UNICAST



MULTICASTING



CLIENT SIDE


Draw GUI
Encapsulate Data
Clean-up Connection
On initialize, draw GUI. [Done only once at the start of the program]
Encapsulate data before sending to server
Encapsulated data.
If the user wants to exit
Client [GUI Thread]
Create the network thread to initialize connection with the server.
Network Thread
Listen on user input, encapsulate it and send data to server accordingly.
Send any received data from server to front-end, accordingly.
Set a completion routine to read on data from server.
Decapsulate Data
Decapsulated data
Client Receive Completion Routine
Decapsulate all data received from the server.

Pseudocode

Server Pseudocode (Unicast Mode)

Send Completion

```
Handle Errors
Check for Cancellation or EOF (finished sending)
If yes
    Free buffer
    Return
Else
    Read data into buffer
    Post WSA Send
```

Receive Completion

```
Handle Errors
If number of bytes received is 0 (received FIN or RST)
    Close socket
    Return
Read Packet
Handle Request
```

Handle Request

```
Read packet type
Switch on packet type
    Case File Request:
        Send File
        Break
    Case Search
        Send Search Results
        Break
    Case Stop Sending
        Set Cancellation Flag
        Break
    Default
        Log Error
        Break
Return packet type
```

Send File

```
Search database for file
If no file found
        Send “File not found” message to client
        Return
Read first part of file into buffer
Send file metatdata
Begin sending file
```

Send Search Results

```
Search database for keywords
Send list of matching files (may be empty)
```

Handle Errors

```
Log the error
Close the socket
```

Server Pseudocode (Multicast Mode)

Read Playlist

```
Open current song file
While the server is running
   Attempt to read from current song file
   If EOF
      Close current song file
      Read metadata for next song
      Open next song file
      Send metadata
      Begin streaming file
   Else
      Transmit file portion
```

CLIENT PSEUDOCODE

GUI THREAD

```
Wait for input or network event
Switch event
   Client selected song
      Write song name to buffer
      Signal network thread to send song request
   Client entered search
      Write keywords to buffer
      Signal network thread to send search request
   Client paused audio
      Signal network thread to stop playing audio
   Client joined multicast stream
____Signal network to join specified stream
   Client started speaking into microphone
      Write microphone data to buffer
      Signal network thread that microphone data is available
   Network wrote metadata to buffer
                  Update metadata (song name, artist, length, etc) in GUI
         Network wrote search results to buffer
            Display search results or “No results found” if none
```

NETWORK THREAD

```
Create socket for default mode with completion routine
Wait for client input event
Handle Errors

If the client entered a search request or file name
         Read data
         Encapsulate Data In Packet
         Send Packet to Server

Else If the client attempted to join a multicast stream
   Clean Up Connection
   Create a new socket with multicast address
   Begin receiving from socket

Else if voice data is available
   Read data
   Encapsulate Data in Packet
   Send data to server

Else if the client disconnected or left multicast
   Clean Up Connection
```

Client Receive Completion Routine

Handle Errors
If the socket has data
    Decapsulate the data
    If it's an error message
        Write it to the data buffer
        Signal the GUI thread to print error
    Else If it's metadata
                Write it to the data buffer
        Signal the GUI thread to update the metadata
    Else If it's search results
        Write it to the buffer
        Signal the GUI thread that search results are available
    Else If it's audio data
        Add to the buffer
        If the buffer is full enough and audio isn't playing, start audio
    Else If it's voice data
        Write to buffer
        Play data

Clean Up Connection

Set "End transmission" flag
If Stop Audio flag is specified
        Cancel playing audio
        Flush audio buffer
If not multicast session
        Send "stop sending" packet to server
Else
    Unregister from multicast session
Close the socket

Decapsulate Data

Read packet type
Read data into buffer
Return packet type

Encapsulate Data

Allocate packet struct
Write packet type to packet struct
Write data buffer to packet struct
Return packet struct

TESTING

Test Number	Name	Description	Tools Used	Results
1	Multicast Music Streaming	Streaming music over a multicast IP to multiple clients.	TBA	TBA
2	Unicast Music Streaming	Streaming music over TCP to a single client.	TBA	TBA
3	Multiple Unicast Streams	Streaming music over TCP to multiple clients.	TBA	TBA
4	Voice Streamed Between 2 Clients	Streaming voice between 2 clients.	TBA	TBA
5	Voice Streamed Between All Clients	Streaming one clients voice to multiple clients.	TBA	TBA
6	Stream Control Functionality	Proper pausing and playing of the stream.	TBA	TBA
7	Music Information	Proper music information is received client side.	TBA	TBA
8	Proper Library Indexing	Music library is properly indexed on server startup.	TBA	TBA
9	Stream Skipping	Proper skipping between the next and previous songs over unicast.	TBA	TBA
10	Clean Audio	Audio received from server to client is free of audio skipping and other issues.	TBA	TBA