

---

# TCP/IP Project

## Design Documentation

**Jaymz Boilard**  
**Jared Hudson**  
**Brendan Neva**  
**Steffen L. Norgren**

BLAW 3600 • BCIT • March 16, 2009

---

## **Application Design & Diagrams** **3**

<b>GUI Mockup</b>	<b>3</b>
<b>Overview</b>	<b>7</b>
<b>Application</b>	<b>8</b>
<b>Client Main</b>	<b>9</b>
<b>Client Download</b>	<b>10</b>
<b>Server Download</b>	<b>11</b>
<b>Non-multicast Server</b>	<b>12</b>
<b>Multicast Server</b>	<b>13</b>
<b>Microphone</b>	<b>14</b>
<b>Client Upload &amp; Server Download (from Client)</b>	<b>15</b>

## **Pseudo-code** **16**

<b>Application</b>	<b>16</b>
<b>Client Main</b>	<b>18</b>
<b>Client Download</b>	<b>19</b>
<b>Server Download</b>	<b>21</b>
<b>Non-multicast Server</b>	<b>22</b>
<b>Multicast Server</b>	<b>23</b>
<b>Microphone</b>	<b>24</b>
<b>Client Upload &amp; Server Download (from Client)</b>	<b>25</b>

# APPLICATION DESIGN & DIAGRAMS

---

## GUI Mockup

---

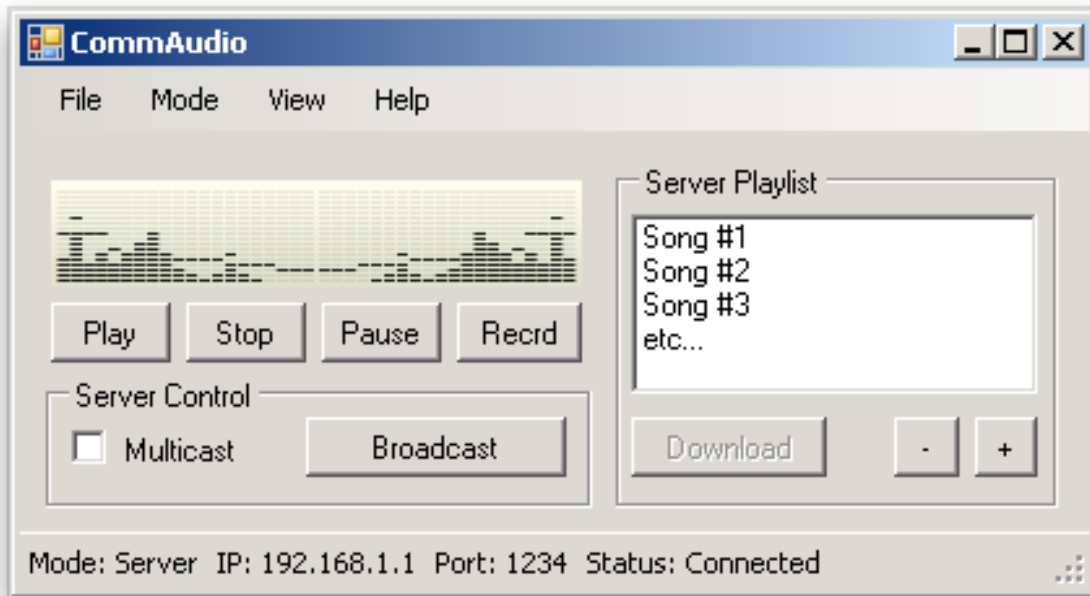


figure 1: Server Mode

We have decided to incorporate the client and server side of this project into a single application. As such, various aspects of the application will be enabled or disabled based on the mode in which the application is currently operating.

The main abilities of the client/server application is the ability to control what is currently streaming through the play/stop/pause functions. Additionally, the record function will cause all currently streaming/multicast sessions to cease and bring the application into recording mode.

All functions are available to the GUI while in server mode except for the download function, as it will always be the client that downloads from the server. Specific functions available to the server are as follows:

- Multicast – Set the server into multicast mode, allowing new client connections until the server begins to broadcast.
- Broadcast – Block all future client connections and begin broadcasting the playlist to currently connected clients.

- Add to Playlist (+) – Opens up the file dialogue box, allowing the user to select files to add to the playlist queue.
- Remove from Playlist (-) – Removes selected files from the playlist.

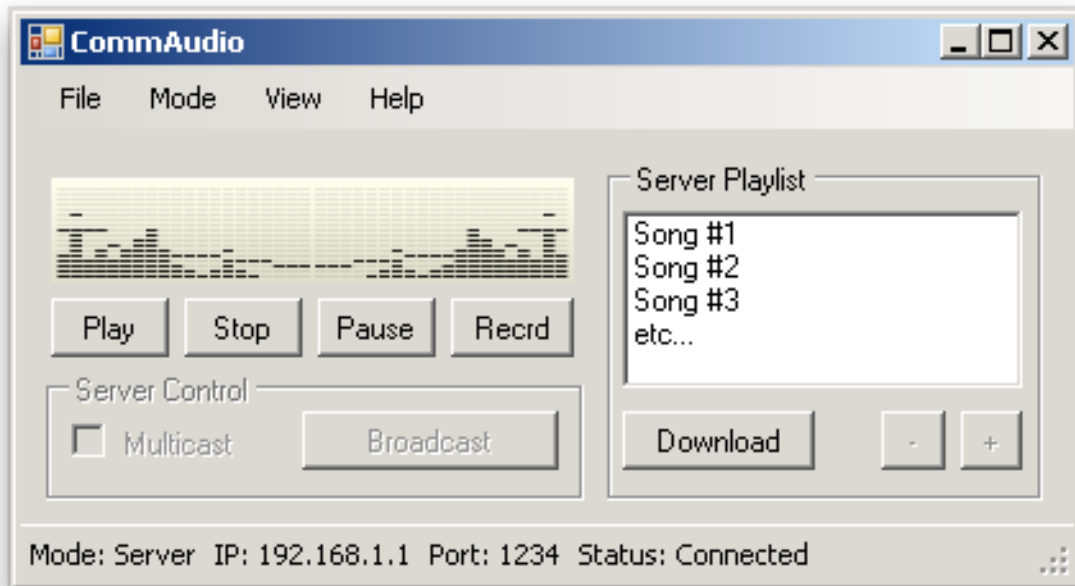


figure 2: Client Mode

While the application is in client mode, all server related functions are disabled, however, the previously disabled “Download” function, is enabled for the client. The download option, however, is only enabled if the client is in streaming mode with the server, not multicast mode. This function allows the client to download any of the selected songs in the playlist.

Additionally, if the client is in streaming mode with the server, the client will have control over which songs in the playlist to play. This is done by selecting the songs in the playlist that the client wants to stream. When the client presses play, it will notify the server which songs to stream to the client.

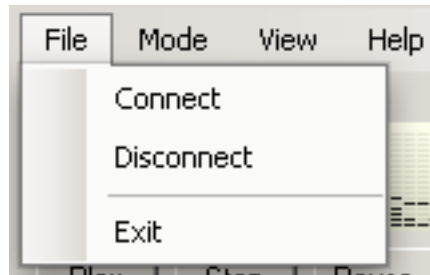


figure 3: File Menu

The file menu allows both the client and server to exit the application. However, the behaviour of the “Connect” and “Disconnect” menu items differ based on the mode for which the application is currently set.

While in server mode, the “Connect” menu item will cause the server to start listening for clients. Meanwhile, the “Disconnect” menu item will cause the server to stop listening for new clients and terminate any open connections.

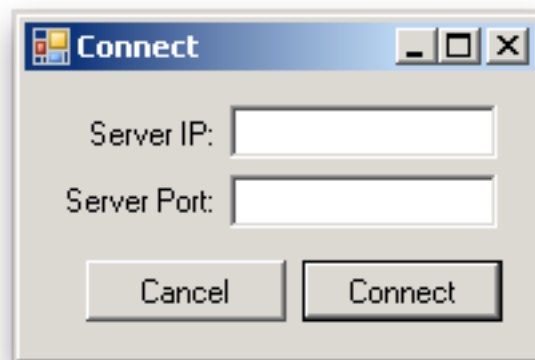


figure 4: Connection Options

If the application is in client mode, the “Connect” menu item will cause a connection options dialog open, prompting the client to enter the server’s IP address as well as the server’s port. By pressing “Connect” the application will first verify the correctness of the IP address and port and then test to see if a server exists at the specified port and IP address. If any of the settings are in error, or the server does not exist, an error message will be presented to the client, allowing the client to re-enter the connection settings and try again.

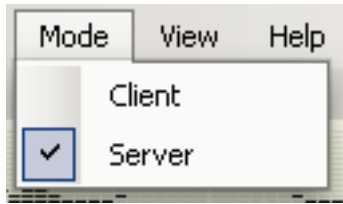


figure 5: Mode Menu

The mode menu allows the user to set whether the application is operating in client or server mode. Changing the application's mode will enable and disable any functions which are not relevant to that mode's operation. Additionally, the user will be unable to change the application's mode while the client or server is connected.

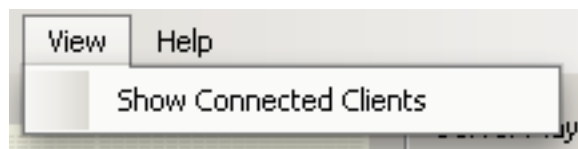


figure 6: View Menu

The view menu allows the user, while in server mode, to view a list of connected clients. This list will appear as a separate window that shows each client's IP address. This list will be updated dynamically, adding and removing clients from the list as they connect and disconnect.

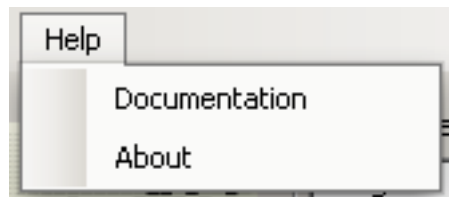


figure 7: Help Menu

The help menu allows the user to view the application's user documentation as well as an about box, which displays the applications version and the developers involved with the application.

The user documentation window will either be a rich text document that is opened through an external program (WordPad) or a built-in dialogue box.

## Overview

---

This is the main overview of how our design documents are organized.

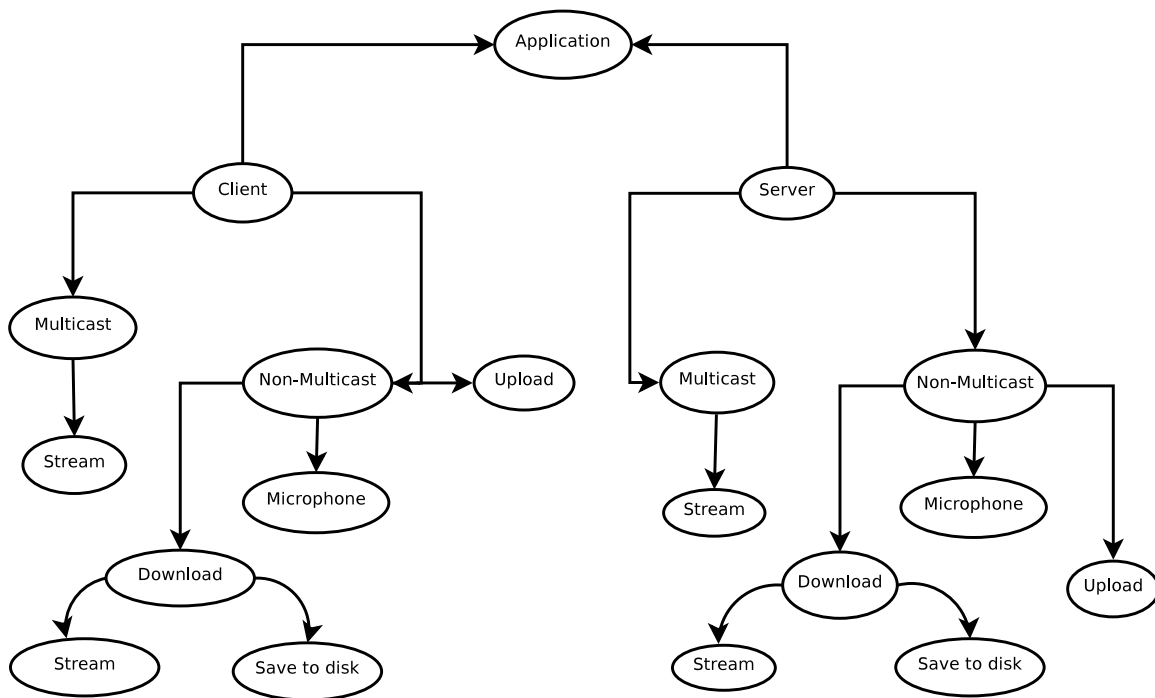


figure 8: Application Overview

When the application starts up you have the option of being a client or server.

If you are the client you can either be a multicast or non-multicast client. Multicast clients can only stream the music being played by the server whereas non-multicast clients can upload, have a microphone conversation or download which consists of streaming or saving the file.

If you decide to be the server you can also be multicast or non-multicast. If the server is multicast all it does is play the songs to the IP's in the multicast IP group. Non-multicast servers can stream a song to a client, have a microphone conversation or download songs from the client. The server-side download portion of the overview symbolizes the server handling how the clients download.

**\*Note:** Wherever a UDP socket is created along with a new service thread, the thread that initialized them goes to the `wndProc` listening for messages.

**\*\*Note:** The rectangles are states and the ellipses are connection points to another diagram.

## Application

This is the main entry point for the program. The program waits for the user to enter the settings which calls the corresponding functions. The server is started first and waits for client connections. If it's the correct type of connection, it will accept, otherwise it will send a busy message.

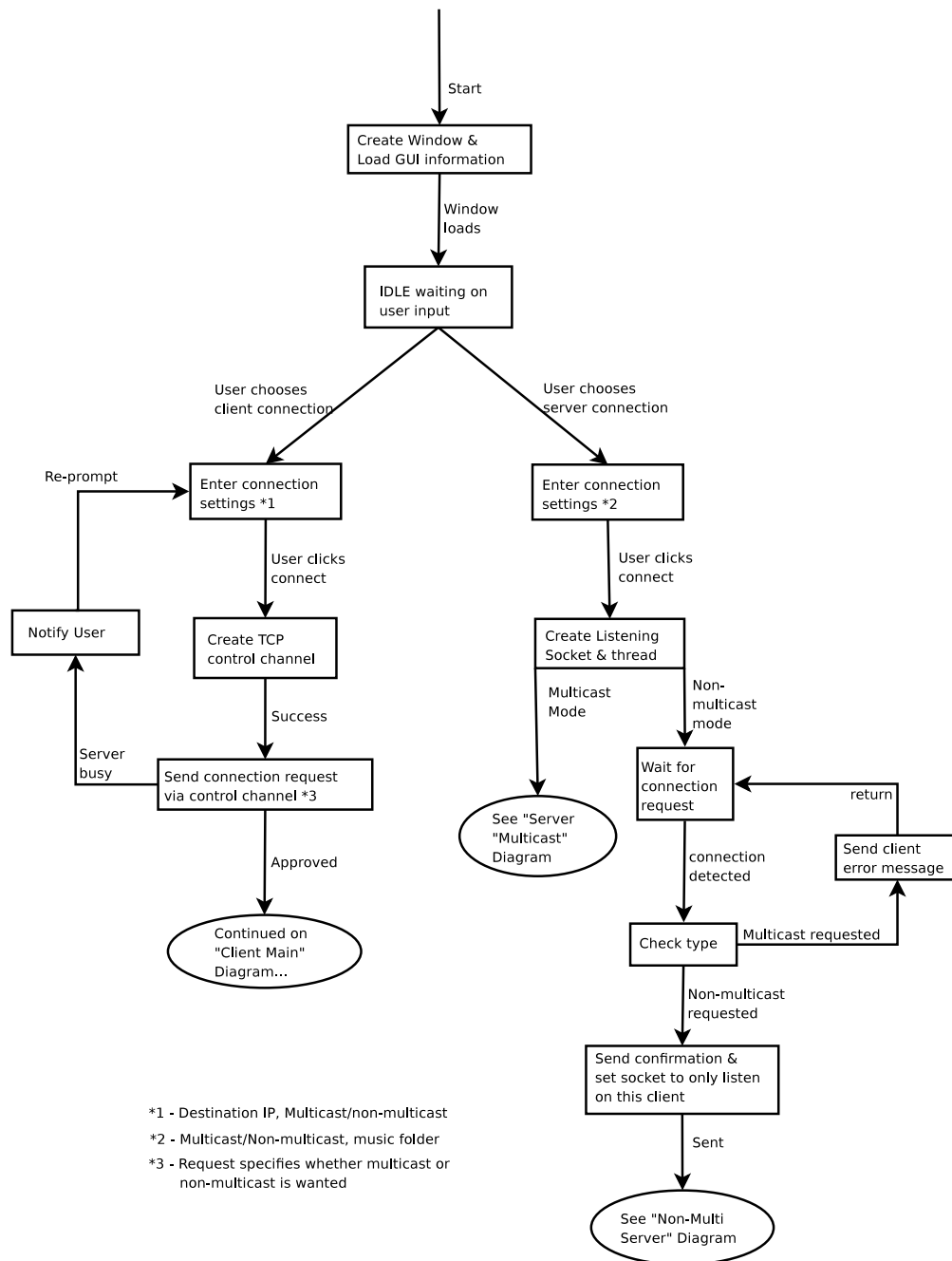
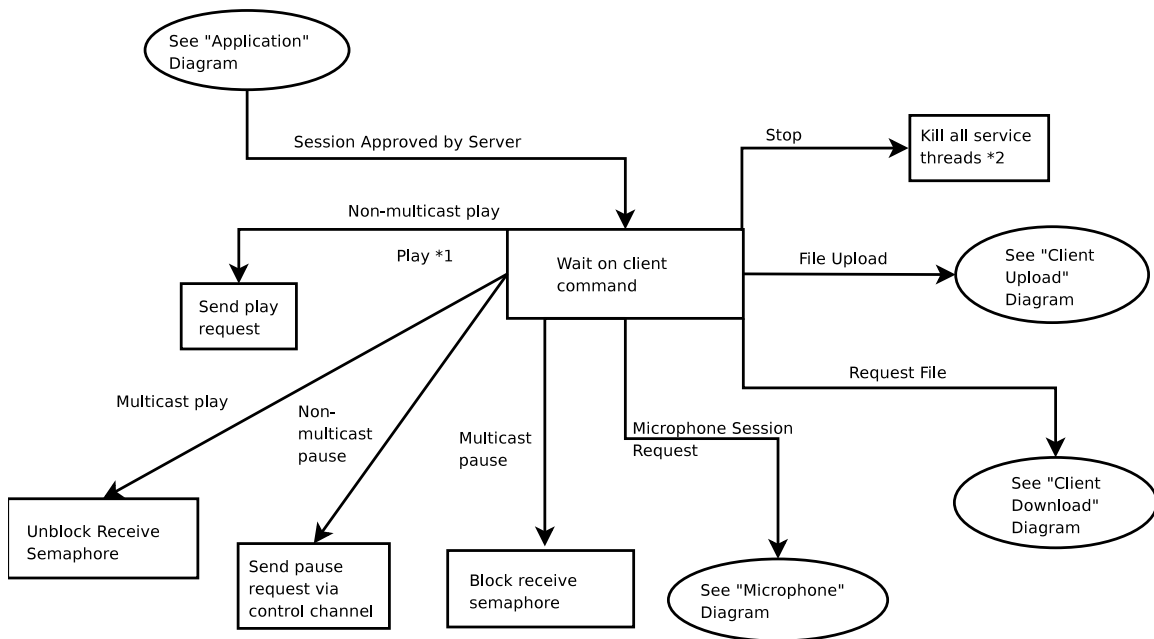


figure 9: Application



## Client Main

The following diagram is how the client deals with the server while in non-multicast mode. The client's input is from what the user selects and is sent to the server via a server control channel.



\*1 - Options only available during a non-multicast audio stream

\*2 - Server will automatically assume that client has pressed stop since it can no longer read from the control channel

figure 12: Client Main

## Client Download

Handles the download for both multicast and non-multicast as well as streaming or save to disk. If it's streaming we use UDP otherwise we use the control channel. The client chooses to download a song, depending if it's multicast or non-multicast determines the outcome. The server will send an EOF message when it is done sending the file(s). If there are multiple files, the server will take out the EOF message between songs until the last one.

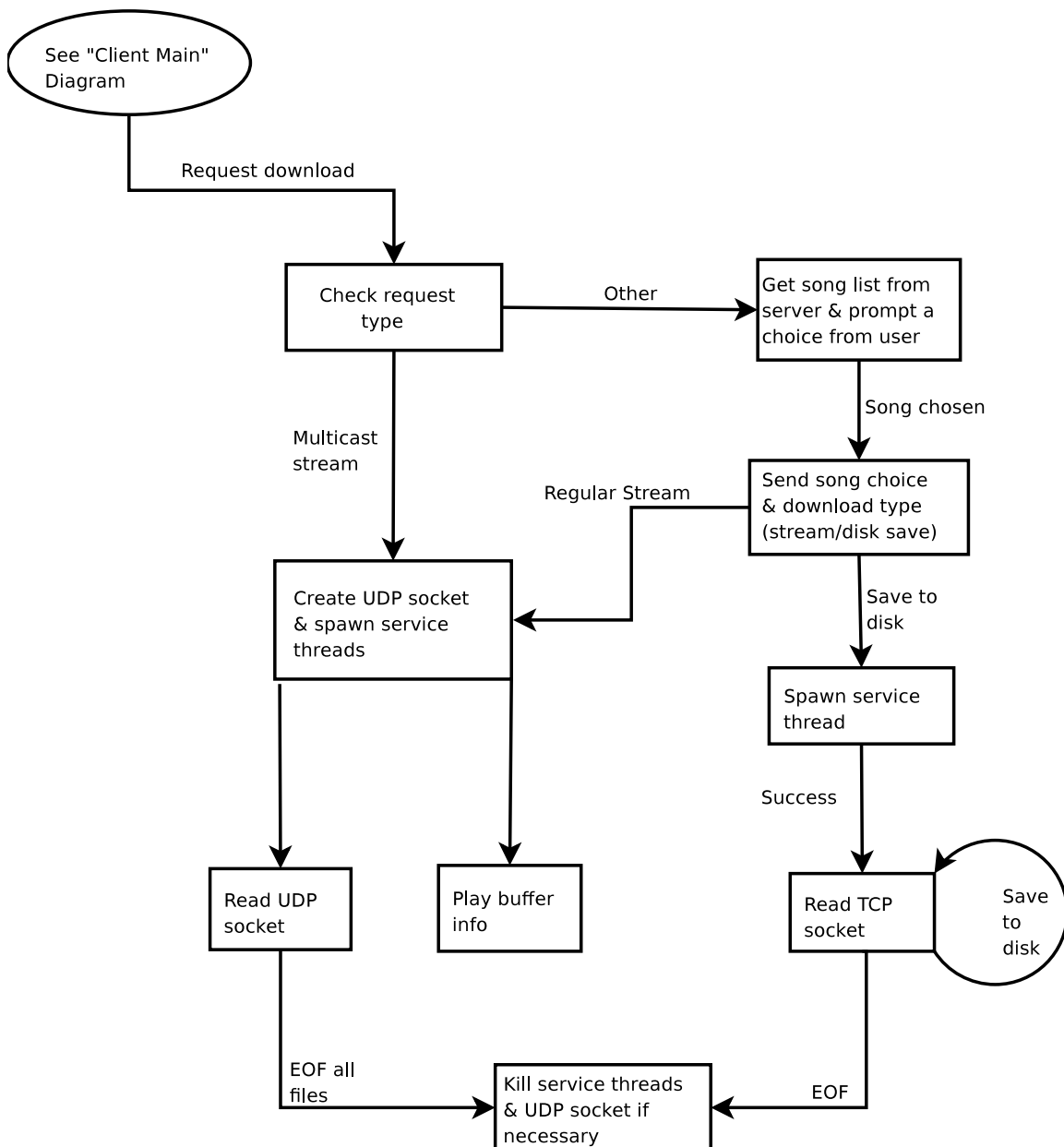


figure 10: Client Download

## Server Download

This diagram shows how the server handles a download request from the client. By the time we get here, the server has already accepted the client connection so there is no need to send a confirmation to the client, acknowledging the request. Instead we just send the song list and wait for the user's song choice and whether they choose to stream or save the file. If stream was chosen then the server will use UDP whereas if save is chosen then it will use the control channel.

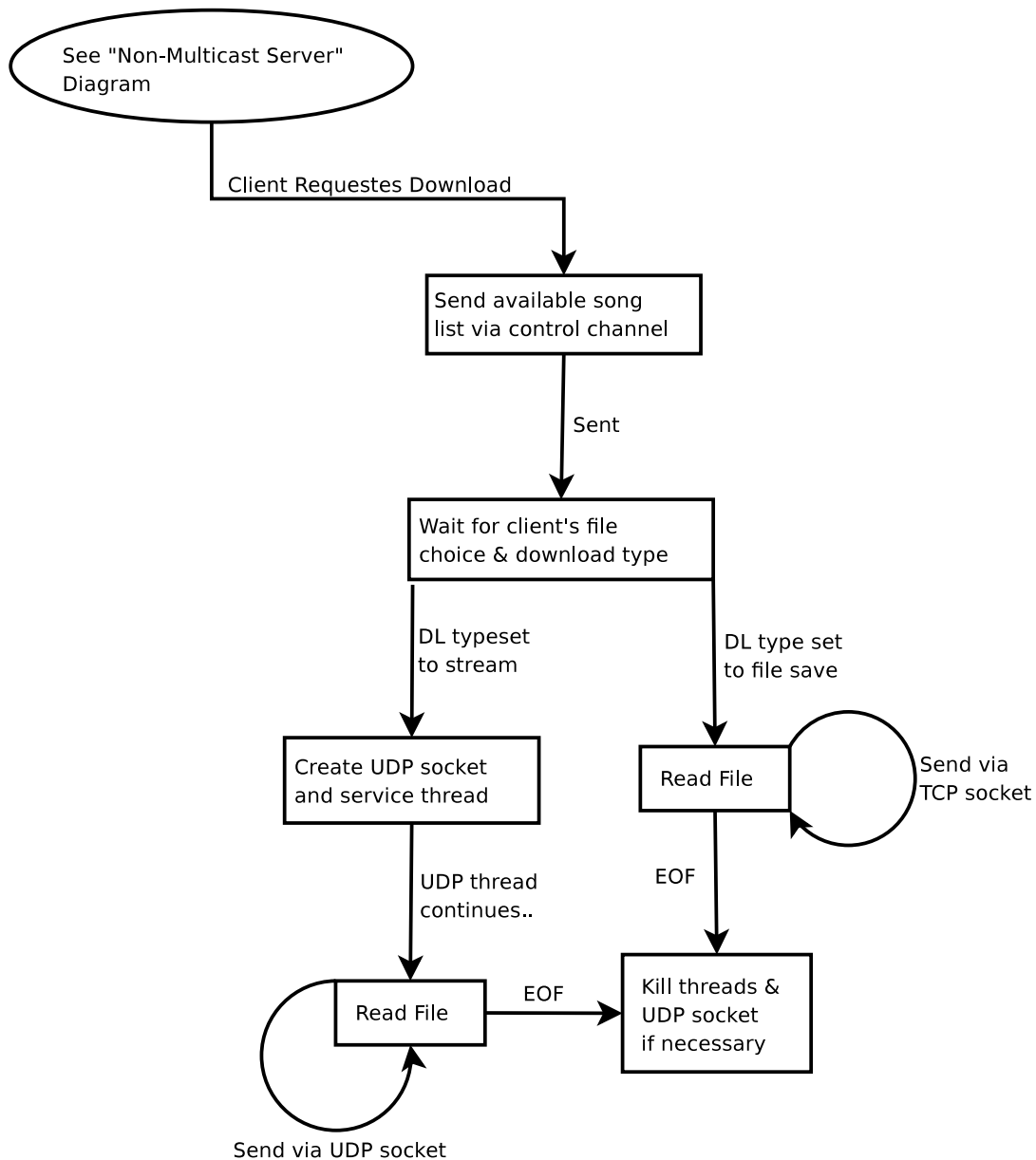


figure 11: Server Download

## Non-multicast Server

This diagram shows how the server responds to client input via the control channel while the server is in streaming mode (non-multicast mode).

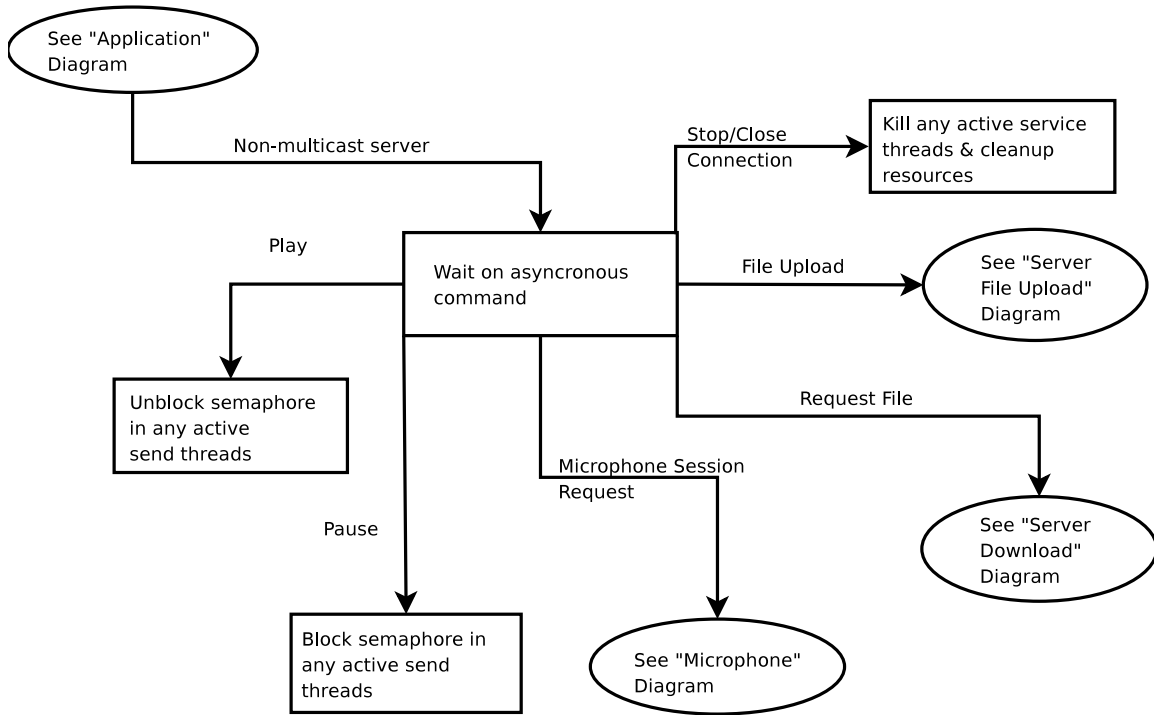


figure 13: Non-multicast Server

## Multicast Server

The server continues to wait for incoming connections in which the client includes its own IP address and connection settings. If the client connection is multicast then the server will add its IP to the multicast group. It will continue to wait for connections until a client clicks the broadcast button which will start the stream. If a client connects while this is happening, it will send a busy message.

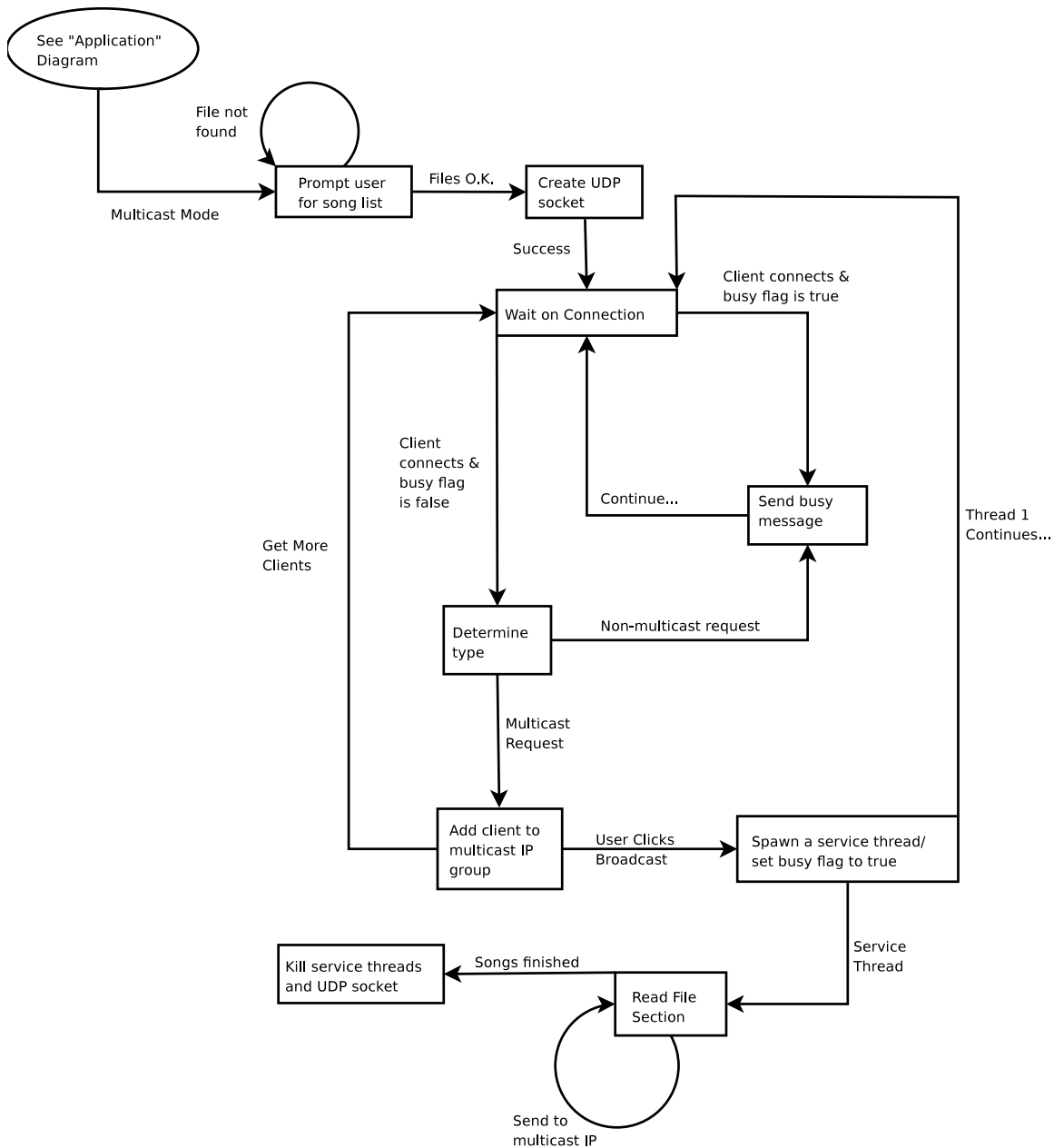


figure 14: Multicast Server

## Microphone

This diagram shows how the client/server works when there is a microphone connection. Since it's a two way connection in which both can send and receive, it will be the same on the client and server.

Note: since the main thread is still getting user input, the send thread will act similar to a multicast server in that when we press pause it will block the receive semaphore on the client. This way the client can prevent feedback caused by sending the data that he/she is receiving through the speakers (noise).

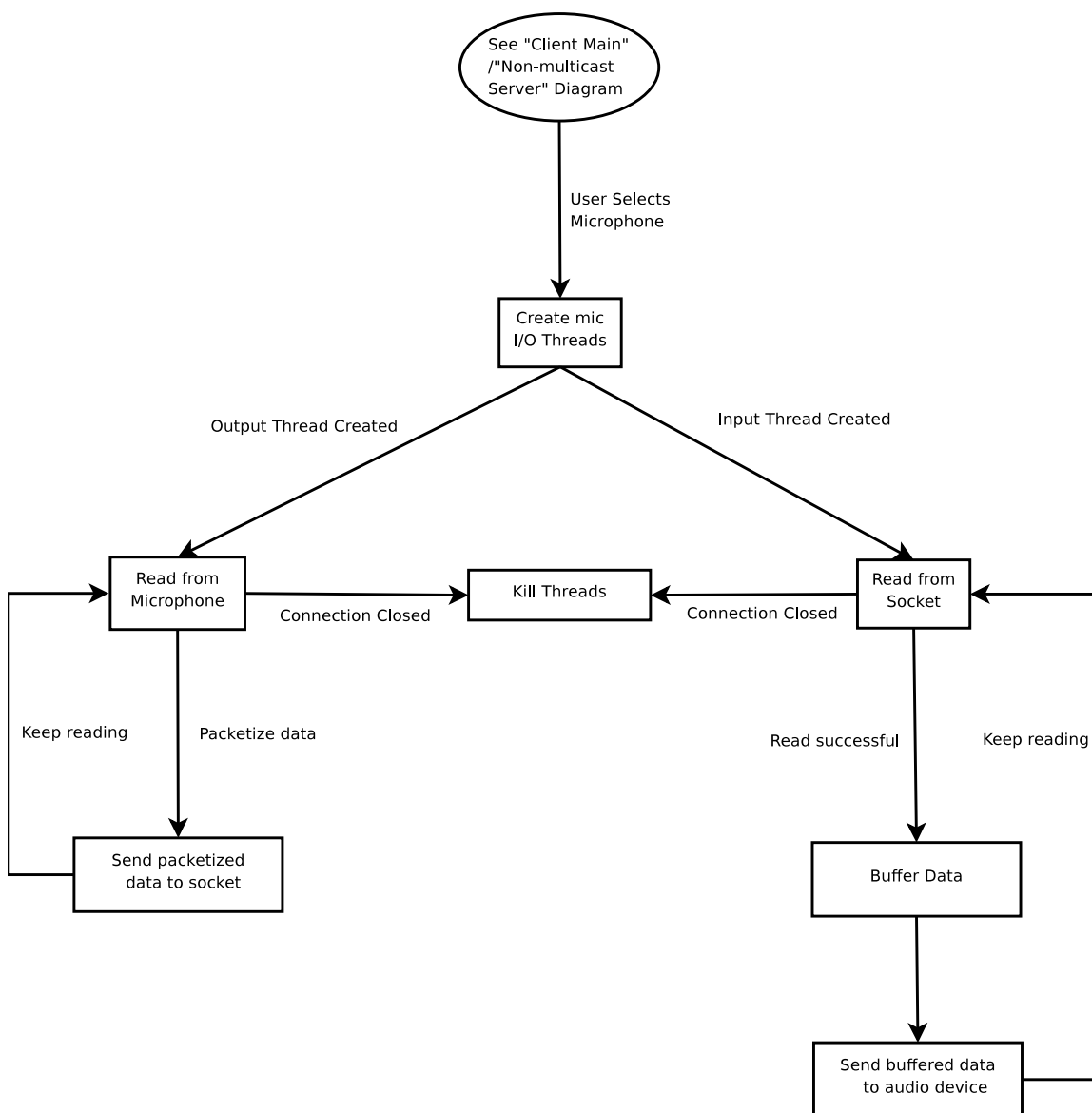


figure 15: Microphone

## Client Upload & Server Download (from Client)

The diagram on the left is how the client handles an upload and the one on the right is how the server handles the client upload.

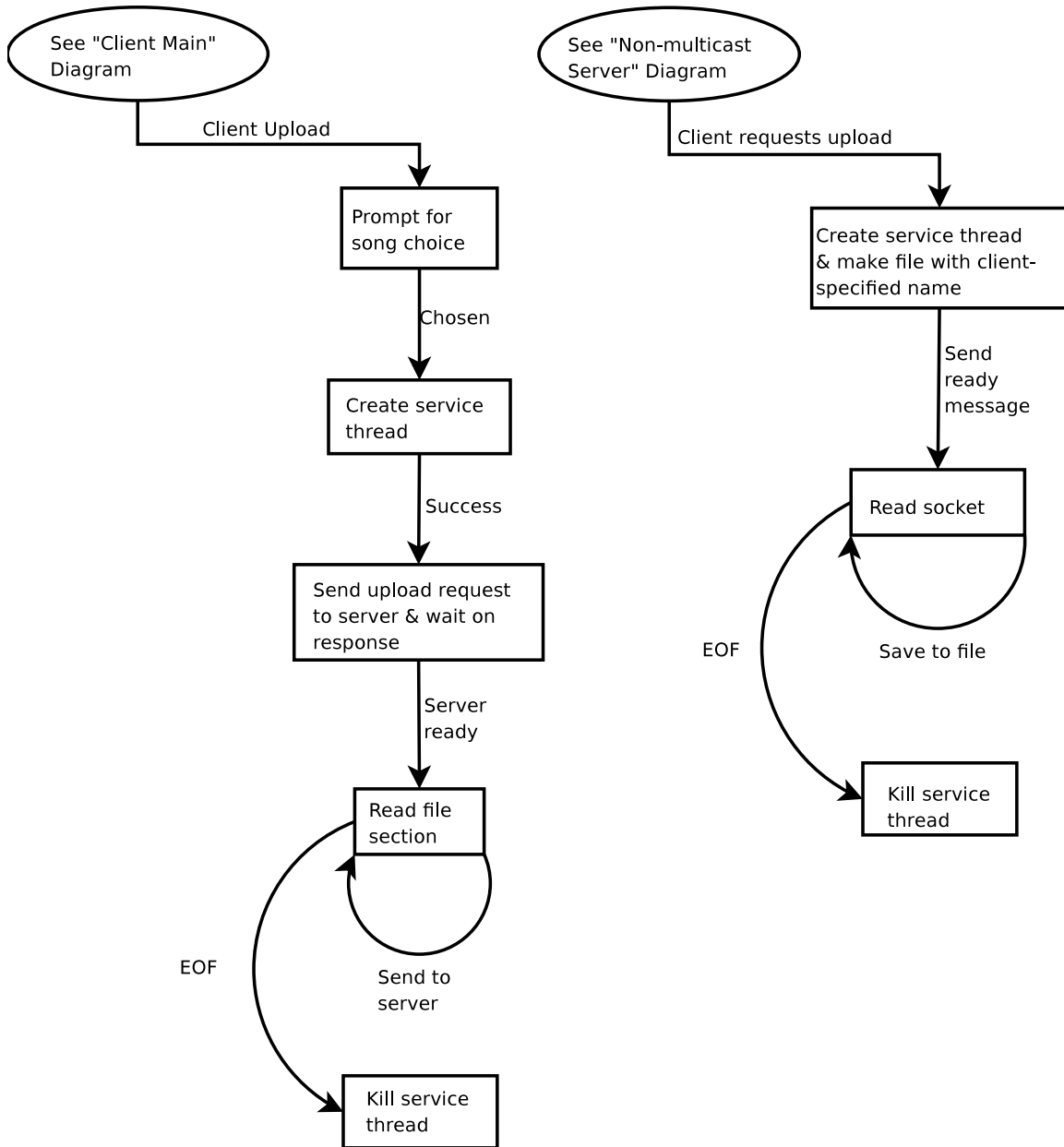


figure 16: Client Upload & Server Download (from Client)

# PSEUDO-CODE

---

## Application

---

### Main start

- set and register the window properties
- create the window
- enter wndProc message loop

### Main end

### wndProc start

allowable messages:

#### WM\_CREATE:

- initialize any data structures necessary for tracking state information
- enable/disable GUI items as necessary

#### WM\_SIZE

- move main window
- move all child windows

#### WM\_PAINT

- redraw the window items & text fields

#### WM\_COMMAND

- call handle\_inputs function

### wndProc end



handle\_inputs start

input messages:

connection choice:

if server, set type to server

if client, set type to client

connect button:

get connection settings from the text fields

if settings are set to multicast server:

call serverMulticast function

if settings are set to non-multicast server:

listen for client connection request

on connection, check the request type:

if muticast requested:

send error message & continue listening

if non-multicast requested:

send confirmation & set socket  
to listen to only this client

call nm\_server function

if settings are set to client:

create a TCP control channel

send a connection request to the server  
via the control channel

if server is busy or connection type doe  
not match:

notify the user and re-prompt  
connection settings

if server approves the session:

call client\_main function

handle\_inputs end

## Client Main

---

### client\_main start

user inputs/requests:

Non-Multicast Play (only available during a stream)

send play request to server via control channel

Non-multicast Pause

send pause request to server via control channel

Multicast Play

unlbock receive semaphore

Multicast Pause

block receive semaphore

File Upload

call client\_upload function

Microphone Session

send microphone request to server via control channel

if accepted

call microphone function

Request File

call client\_download function

Stop

kill all service threads & UDP socket (if exists)

### client\_main end

## Client Download

---

### client\_download start

if request is multicast:

    call udp\_services function

if request is non-multicast:

    get song list from the server

    prompt the user to choose a song from the list

    send the server our song choice & download type (TCP/UDP)

    if we want to stream (UDP):

        call udp\_services function

    if we want to save to disk (TCP):

        create a service thread

        service thread:

            read the TCP socket, storing it to a buffer

            save buffer data to file

            if EOF is found:

                kill service thread

### client\_download end

udp\_services start

create UDP socket

create service threads

thread 1:

play information stored in buffer

thread 2:

read UDP socket & store in buffer

if we have read to EOF on all files

kill service threads

kill UDP socket

udp\_services end

## Server Download

---

### server\_download start

```
Send song list via control channel
wait for response
if response is 'stream'
    create stream thread
if response is 'download'
    create save thread
```

### server\_download end

### Stream thread start

```
create & bind UDP socket
loop reading contents from file
    store in buffer
    send buffer to UDP socket
if EOF
    close socket
```

### Stream thread end

### Save thread start

```
create & bind TCP socket
loop reading contents from file
    store in buffer
    send buffer to TCP socket
if EOF
    close socket
```

### Save thread end

## Non-multicast Server

---

### nm\_server start

client requests:

Play

unblock send semaphore

Pause

block send semaphore

Stop/Close connectoin

kill all send threads

cleanup resources (such as sockets)

Request File

call server\_download function

File Upload

call server\_upload function

Microphone

call microphone function

### nm\_server end

## Multicast Server

---

### serverMulticast start

prompt user for song list

if the file isn't found:

re-prompt

create UDP socket

wait for a client to connect until the user clicks broadcast

client connects and busy flag is set to true:

send a busy message to that client

client connects and busy flag is set to false:

if client's request is for a non-multicast session

send a busy message to that client

if client's request is for a multicast session

add the client's IP to the multicast IP group

continue getting more clients

set the busy flag to true

spawn a service thread

loop copying sections of file to the buffer

send buffer information to the multicast IP group

when all songs are finished:

kill service thread

close UDP socket & resources

### serverMulticast end

## Microphone

---

microphone start

    Create input thread

    Create output thread

microphone end

input thread start

    Open socket

    if open successful

        read while connection active

            add data to buffer

                play buffer

input thread end

output thread start

    Open socket

    If open successful

        Read while play button active

            Packetize buffer

                Send packetized buffer to socket

output thread end



## Client Upload & Server Download (from Client)

---

### server\_upload start

```
create file with specified name
create service thread
send ready message
while not EOF
    read socket data and save to the file
kill service thread
```

### server\_upload end

### client\_upload start

```
create service thread
prompt for file choice
create service thread
send upload request //wait for response

while not EOF
    read file & send data over control channel //TCP
kill service thread
```

### client\_upload end