Comm Audio

Design Documentation

Jaymz Boilard
Jerrod Hudson
Brendan Neva
Steffen L. Norgren

COMP 4985 • BCIT • April 14, 2009

TABLE OF CONTENTS

Application Design & Diagrams

GUI Mockup

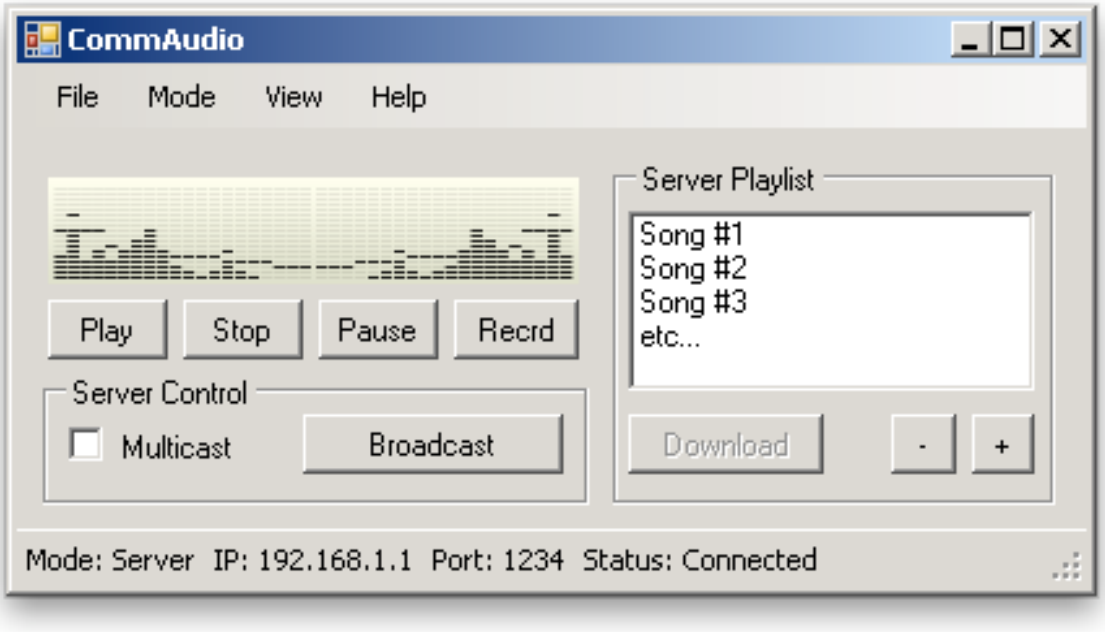


figure 1: Server Mode

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.

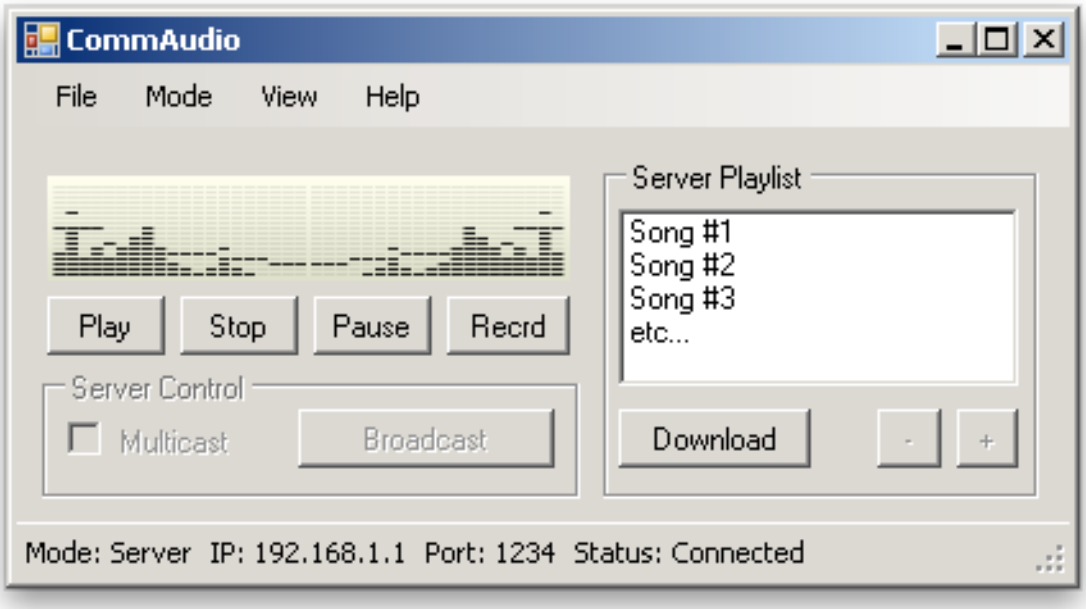


figure 2: Client Mode

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.

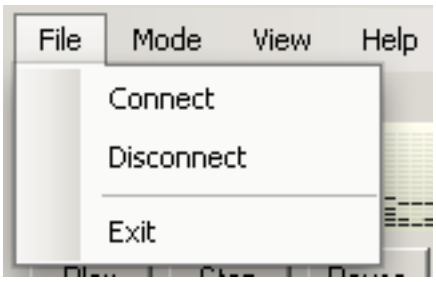


figure 3: File Menu

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.

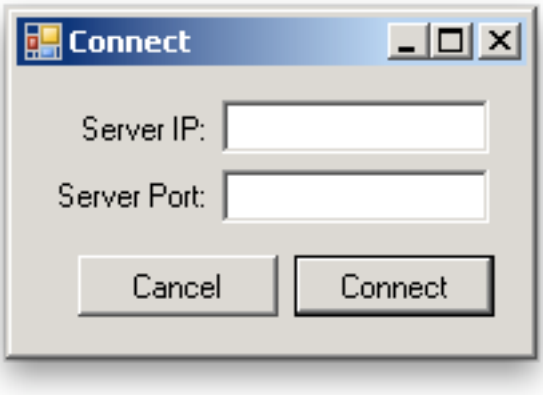


figure 4: Connection Options

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.

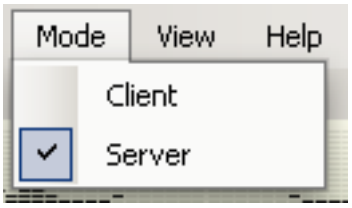


figure 5: Mode Menu

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.

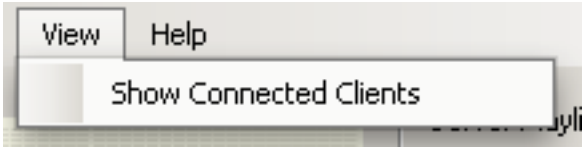


figure 6: View Menu

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.

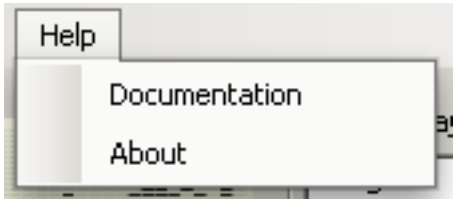


figure 7: Help Menu

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.

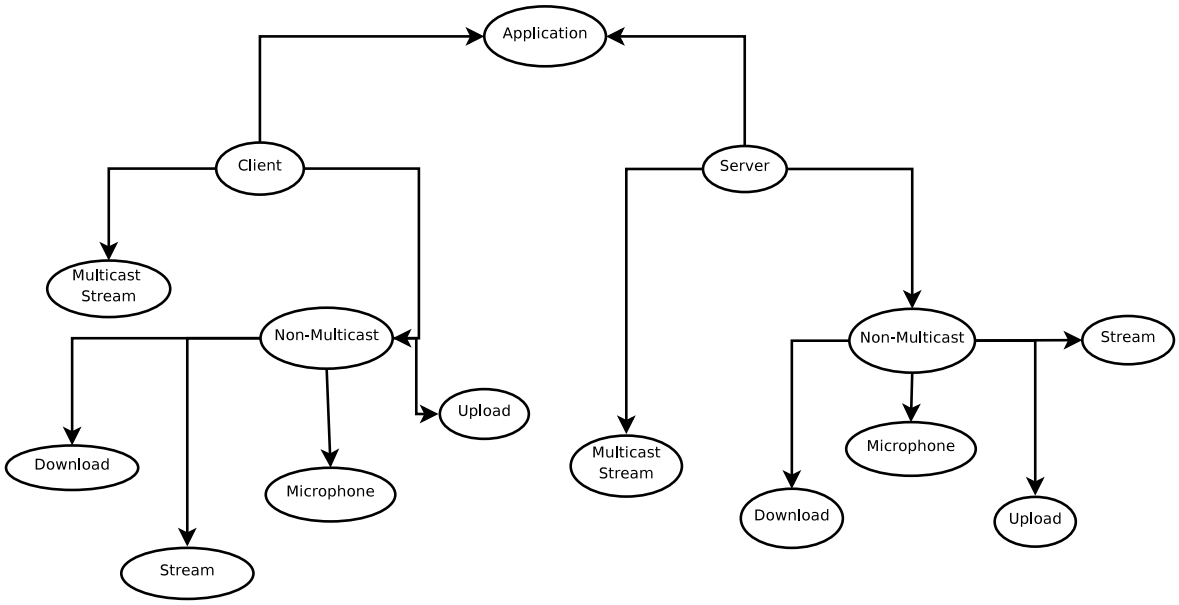


figure 8: Application Overview

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.

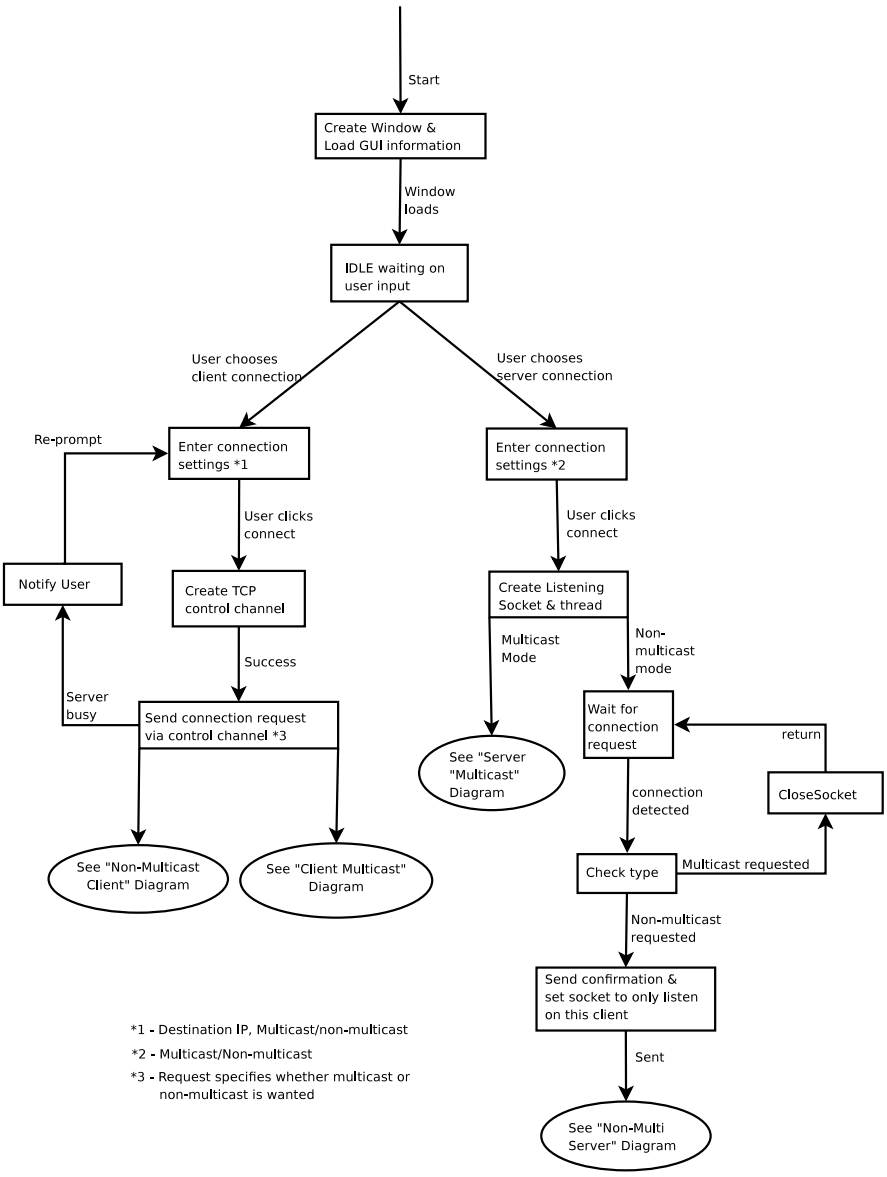
If you decide to be the server you can also be multicast or non-multicast. If the server is multicast all it does is play the songs to the IP's in the multicast IP group. Non-multicast servers can stream a song to a client, have a microphone conversation or download songs from the client. The server-side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

***Note:** Wherever a UDP socket is created a long 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

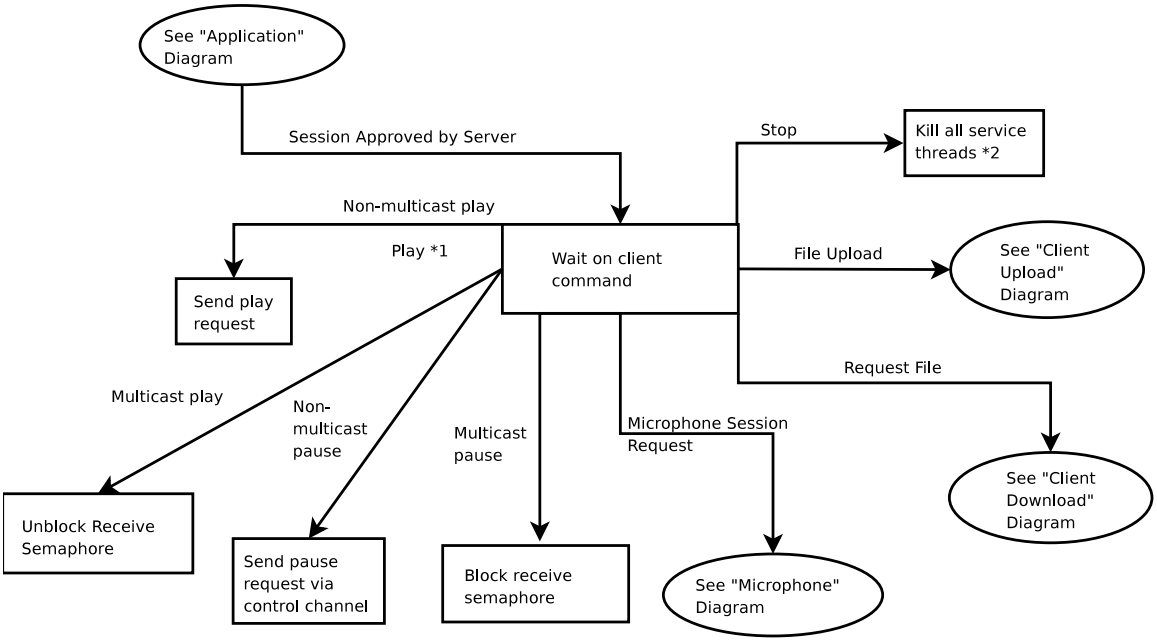


client connections. If it's the correct type of connection, it will accept, otherwise it will send a busy message.

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.

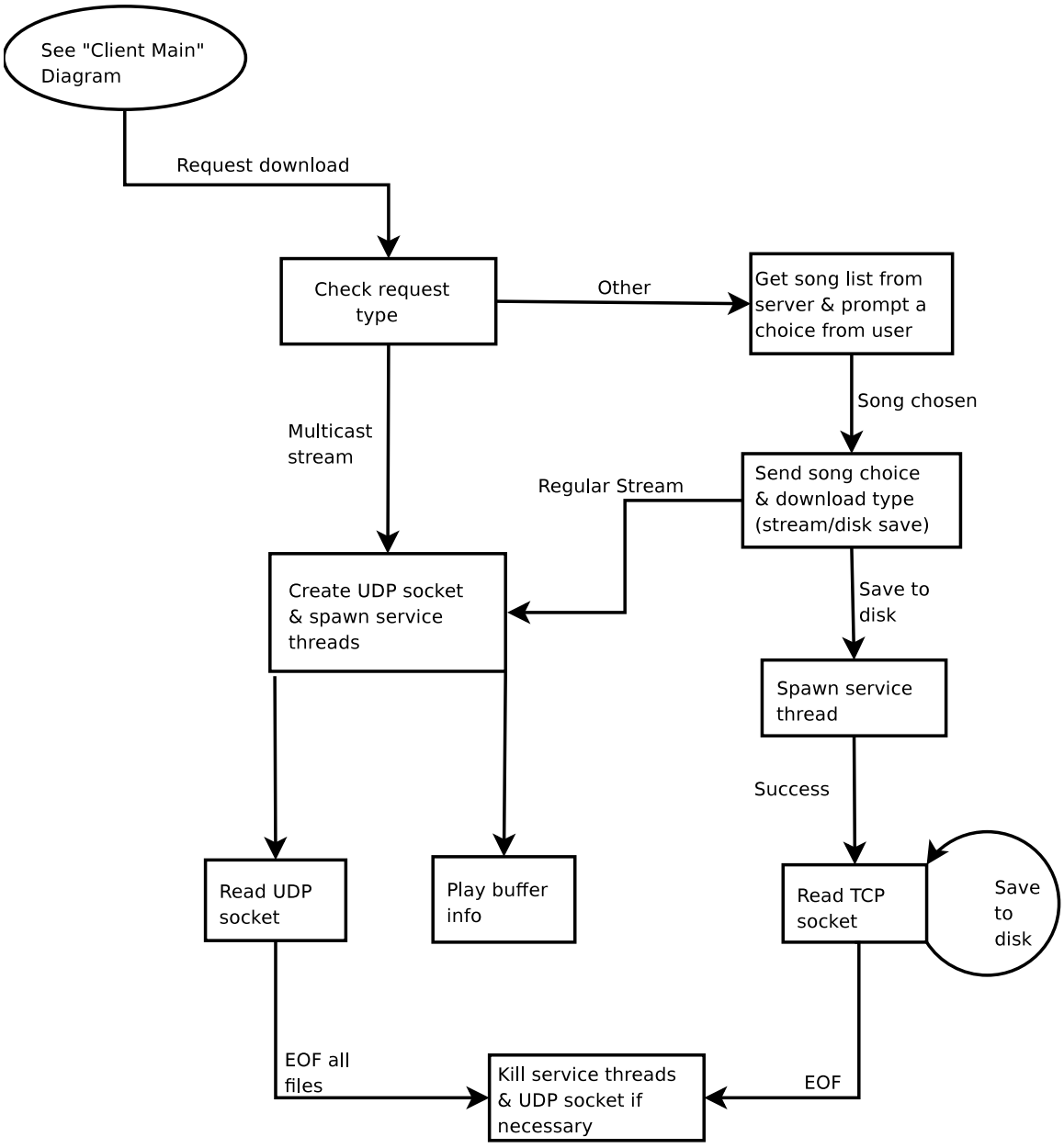


figure 10: Client Download

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.

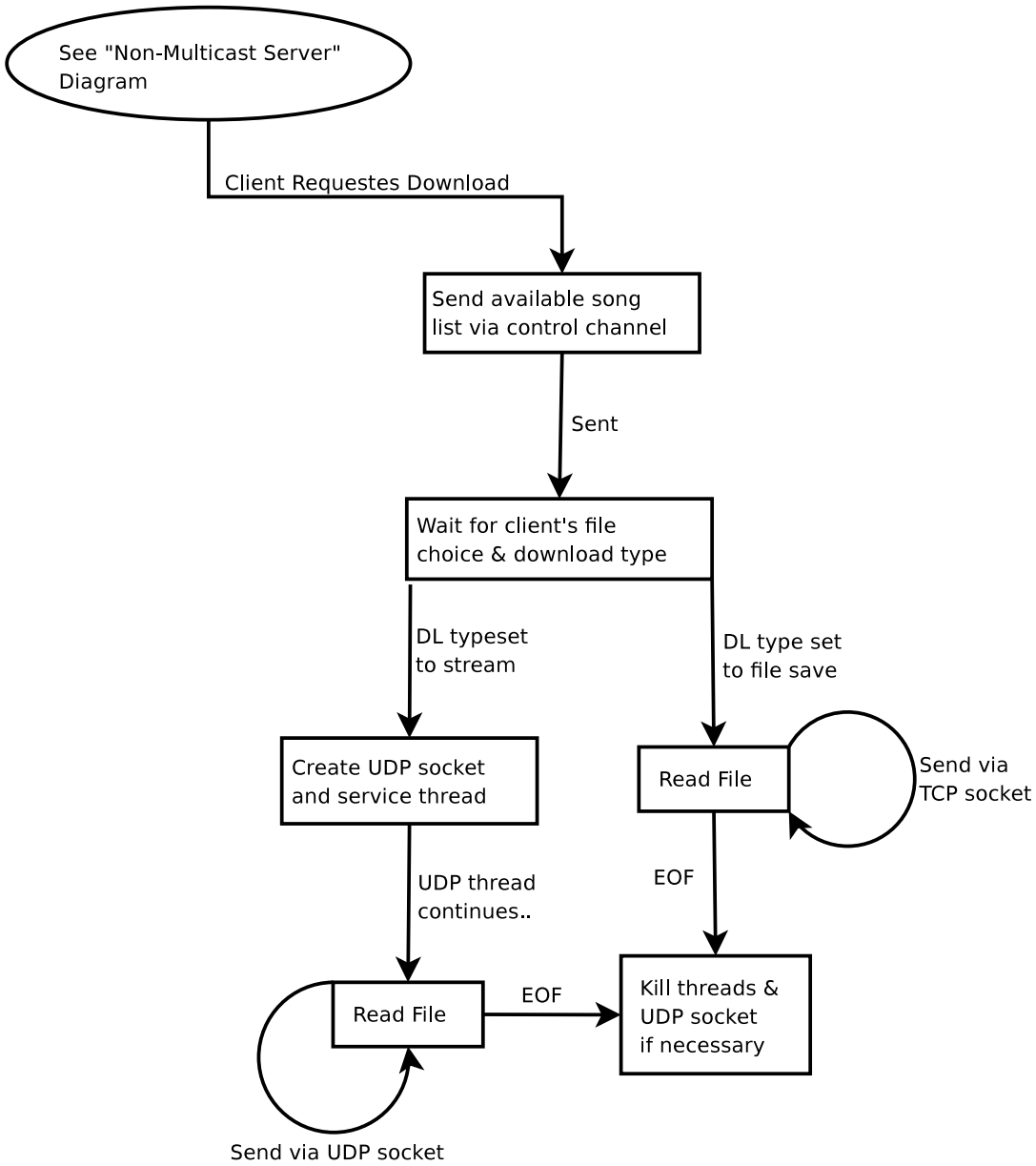


figure 11: Server Download

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).

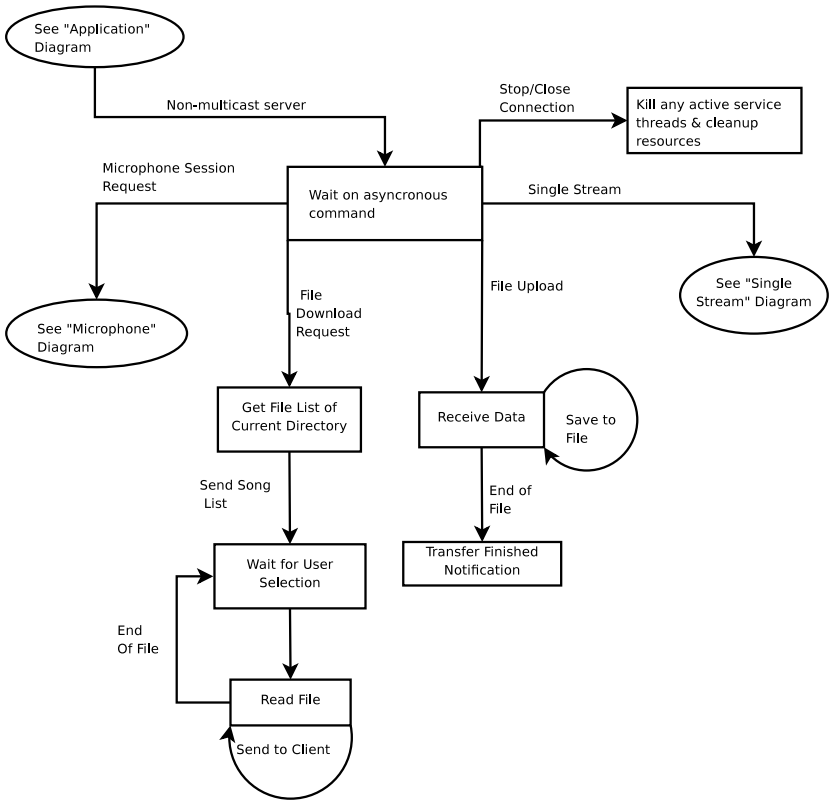


figure 13: Non-multicast Server

Non-Multicast Main

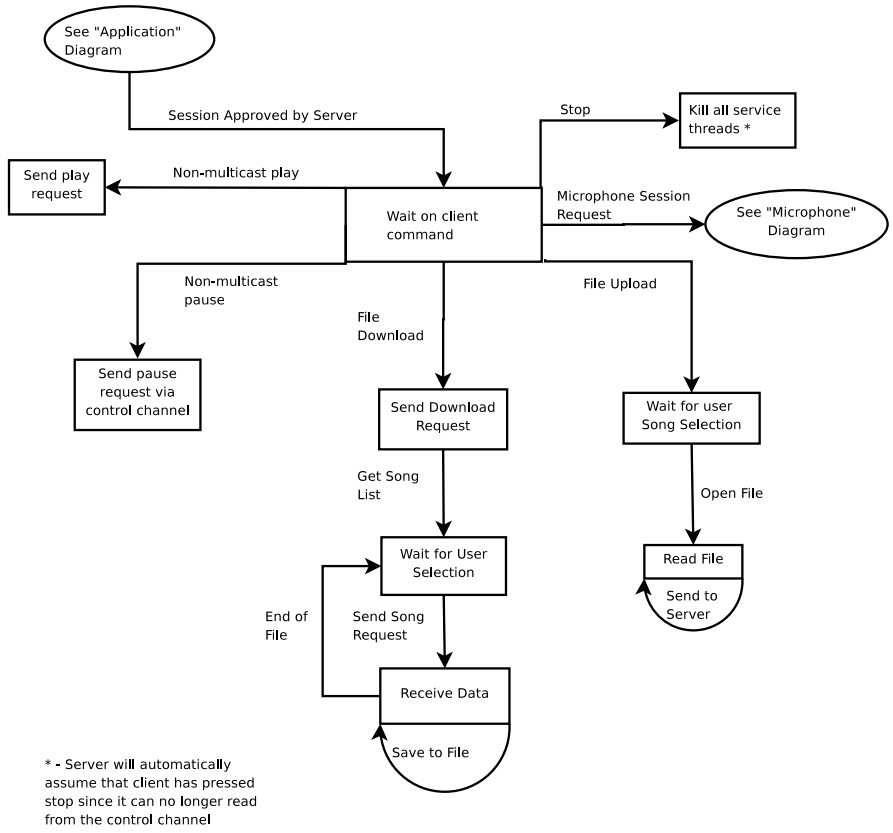


figure 14: Multicast Server

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).

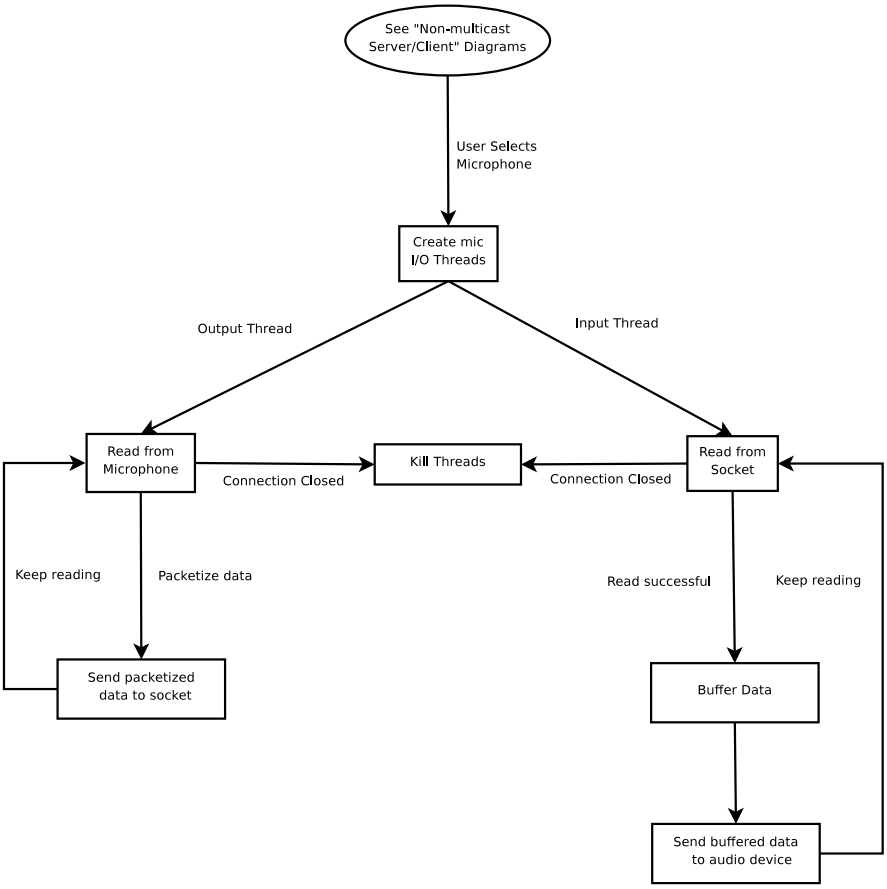


figure 15: Microphone

Client & Server Multicast

These diagrams show how the client and server manage multicasting.

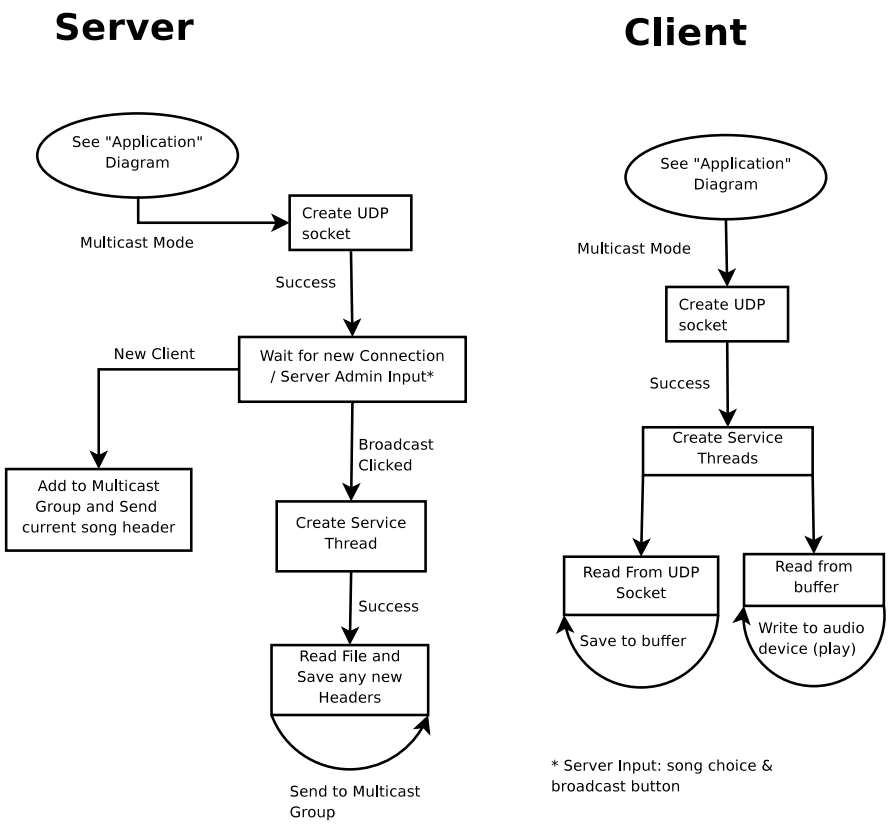


figure 16: Client Upload & Server Download (from Client)

PSEUDO-CODE

Application

```
OnCommand start
     case FILE_CONNECT
          call setup_server(tcpSocket)
          call setup_server (udpSocket)
     case FILE_DISCONNECT
          sockClose
          Enable/Check menu items
          Update status bar (disconnected)
     case FILE_EXIT
          PostQuitMessage
     case CLIENT
          Update CI Struct to Client
          create Socket
          enable Menu Items
     case SERVER
          Update CI Struct to Server
          Create Socket
          EnableMenu items
     case SINGLE_UPLOAD
          Update CI Struct
          CheckMenu items
     case SINGLE_STREAM
          Update CI Struct
          CheckMenu items
```

```
    case MULTI_STREAM
        Update CI Struct
        CheckMenu items

    case FILE_LOCAL
        check Busy flag
        if OK set busy flag to true
        Browse files
OnCommand end
```

Non-Multicast Client

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
          File Upload
               call client_upload function
          Microphone Session
               send microphone request to server via control channel
               if accepted
                    call microphone function
          File Download
               call client_download function
          File Upload
               call server_download function
          Stop
               kill all service threads & UDP socket (if exists)
```

client_main end

Non-multicast Server

nm_server start

client requests:

Stop/Close connection

kill all send threads

cleanup resources (such as sockets)

Single Stream

call sendStream function

File Upload

call server_download function

File Download

call client_download function

Microphone

call microphone function

nm_server end

Client Download

```
client_download start
     get song list from the server
     send the server our song choice
     while receiving song data
          write to a file
client_download end
```

Server Download

```
server_download start
     User selects song
     while not end of file
          send song data to server
server_download end
```

Multicast Server

```
serverMulticast start
     create UDP socket
     Wait for new Connection/Broadcast button
     If new client
          add to Multicast Group
     If Broadcast pushed
          Create Thread
          Read file and save new headers
          send to Multicast group
serverMulticast end
```

Microphone

sendMicrophone start

```
if Microphone
    call mic_record_begin function
    Create Thread for receiveMicThread
```

sendMicrophone end

receiveMicThread start

```
allocate data blocks
receive data via UDP
call writeAudio function
```

receiveMicThread end

mic_record_begin start

```
allocate buffers and wave headers
Initialize structs
Open the input audio device
```

mic_record_begin end

Initial Setup/Helper Functions

```
Main start
     set and register the window properties
     create the window
     enter wndProc message loop
Main end


wndProc start
     switch messages:
          HANDLE_MSG(WM_CREATE)
               call OnCreate

          HANDLE_MSG(WM_SIZE)
               call OnSize

          HANDLE_MSG(WM_COMMAND)
               call OnCommand

          HANDLE_MSG(WM_PAINT)
               call OnPaint

          HANDLE_MSG(WM_CLOSE)
               call OnClose

          HANDLE_MSG(WM_SOCKET)
               call OnSocket
```

```
          HANDLE_MSG(WM_DESTROY
               call OnDestroy
wndProc end

OnCreate start
     Initialize window
     Setup CONNECTION_INFO struct
OnCreate 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

OnPaint start
     Begin/end paint
OnPaint end
OnDestroy start
     PostQuitMessage
OnDestroy end
```

```
OnSocket start
     switch WSAEvent
          case FD_ACCEPT
               sockAccept()
          case FD_CLOSE
               sockClose()
          case FD_CONNECT
               sockConnect()
          case FD_READ
               tcp_sockRead()
          case FD_WRITE
               writeTCPsock()
OnSocket end
```

Socket Functions

sockAccept start

SOCKET s

s = accept()

sockAccept end

sockClose start

close socket

sockClose end

sockConnect start

error checking connection

-already connected

-couldn’t connect

-timed out

sockConnect end

tcp_sockRead start

if Server

if Single Upload

call client_download

recv() first packet to determine type

if Single Download

call server_download

if Single Stream

Check if previous stream thread is open

```
                call sendStream thread
          if Microphone
                ****************************************
          Set request type


     if Client
          if Single Download
                call client_download
          if Single Stream
                get Song listing
          if Microphone
                ****************************************
```

tcp_sockRead end

writeTCPsock start

```
     clear the buffer
     Set request type
     Send request type


     if Single Upload
          call server_download
     if Microphone
          ****************************************
```

writeTCPsock end

Local Play Functions

```
localSong_Init start
     initialize DeviceType to “waveaudio”
     initialize ElementName to the file
     if Opening the device for file input fails
          error
     else
          set the DeviceID to proper device
          Send Play message to the device
localSong_Init end

localSong_Pause start
     if DeviceID is ok
          Send pause message to device
localSong_Pause end

localSong_Play start
     if DeviceID is ok
          send Play message to device
localSong_Play end

localSong_Stop start
     if DeviceID is ok
          Send stop message
          Send close message
localSong_Stop end
```

Setup Client

setup_client start

WSAstartup();

create TCPSocket/UDPSocket

set SO_REUSEADDR so we can reuse the port

WSAAsyncSelect on CONNECT, WRITE, READ and CLOSE

Initialize SOCKADDR_IN struct

setup_client end

Setup Server

setup_server start

WSAStartup()

create TCPSocket

Set SO_REUSADDR so we can reuse port

initialize SOCKADDR_IN struct

bind the address to the socket

Listen for 5 connections

WSAASyncSelect on READ, ACCEPT and CLOSE

setup_server end

Streaming (Multicast/Single Client) Functions

receiveStream start

```
allocate block variables
Initialize Critical Section
While receiving
    if Single Stream
        send Play signal to server
    receive data from server
    if buffer contains header
        update the waveform header
        reopen the audio device
    call writeAudio function
Wait for audio to complete
free the allocated blocks
delete critical section
```

recieveStream end

sendStream start

```
Open the file to be sent
if Single Stream
    wait for initial UDP packet to determine address
While not EOF
    Read the file
```

```
            if Multicast
                  if last file chunk read contained a header
                        save the header information
                  if new client connects
                        send header to client
            send data to client
            if Single Stream
                  wait for next packet request from client
            if Multicast
                  wait for audio to finish playing
sendStream end

writeAudio start
      while there is data to process
            if header is already prepared
                  unprepare header
            prepare new header
            play the sound
writeAudio end
```