|  |
| --- |
| COMP 4985 Assignment 3 Data Communications 2013 |
| Design Document |
|  |

|  |
| --- |
| Jesse Wright, John Payment, Luke Tao  4/2/2013 |

Table of Contents

[Overview 2](#_Toc352680892)

[Client 3](#_Toc352680893)

[State Machine 3](#_Toc352680894)

[Server 4](#_Toc352680895)

[State Machine 4](#_Toc352680896)

[Pseudo Code 5](#_Toc352680897)

[Multicast Set Up 7](#_Toc352680898)

[State Machine 7](#_Toc352680899)

[Pseudo Code 8](#_Toc352680900)

[TCP File Transfer 11](#_Toc352680901)

[Flowchart 11](#_Toc352680902)

[Pseudo Code 12](#_Toc352680903)

[UDP P2P Microphone 14](#_Toc352680904)

[Flowchart 14](#_Toc352680905)

[Pseudo Code 15](#_Toc352680906)

## Overview

Buffer Size

- 10KB

Packet Size

- 1024B

Packet Features

- Song Name (limited to 20chars)

~~- Song Length~~

~~- Current Song Progress (IE. How much of the song has played)~~

Stream Speed

- Sending as fast as possible

## Client

### State Machine



## Server

### State Machine



Pseudo Code

**Set up UDP Settings:**

To set up the server for the Multicasting we will firstly need to initialize winsock so that our socket calls will work as they should. Once this is done, we can now create our data gram socket. Next, we will want to set the Multicast Address (hardcoded for now), and the Port(hardcoded for now) so that we can send data to the clients via the multicast address. We now need to set the Multicast Interval setting so we know how long to wait before sending more data. (We will need to do some testing to find an optimal value.) Now, set the time to live setting for the multicast (how many router jumps are made). Essentially this should be no greater than two. Once we have these settings applied we can now bind the socket and make sure to allow for any hosts to connect on any port. We are now ready to join the multicast server.

**Joining the Multicast Server**

Here all we need to do is make sure to set our multicast address to the once specified earlier and allow any others to join as well. Now we just set the socket options to finalize that we are being added to the multicast address. Once this is complete we are a part of the multicast group, but we still need to disable loopback so we aren't getting our own messages that we are sending to the clients. After, the loopback has been disabled we will want to set our destination address to the Multicast address so that it will be broadcasted to others in the group. We are now ready to send data.

**Reading/Sending File**

Here we will be using basic File I/O. We will open the first audio file in our list and start to read from it. Once we have read data from the file **(Specified later)** will send it to our multicast group, wait the amount of time specified by the interval and repeat this process.

**Setting up TCP Settings**

At the same time as the Multicast settings are being applied, we will have to set up a TCP connection for peer to peer data transferring. Firstly, we need to create a listen socket and listen for incoming connections from clients. We need to make sure that we set our socket to be able to listen for any hostname, and also specify the port we will be listening on.

**Wait for TCP Connections**

Once these two things are done we can now bind the socket and start listening for connections. Once a client connection is received we will create a new socket and the client will be dealt with elsewhere so we can listen for more clients.

**Process Request from Client**

Once we have established a connection with the client we will wait for a command from them. This may be in the form on a number (to be determined) but in any case this command will tell us what the client wants to do. Once we have received the request we will do one of two things: a file transfer or a microphone session.

**Microphone Session Selected**

If the client selects a microphone session we will need to repeat the steps in creating a UDP socket but this time we won't be joining a multicast, we will just be going for a peer to peer connection (not really in UDP).

**Monitor for data from client/Process data**

Here we will be waiting for data to arrive on the socket from the client. Once we receive data we will be using a buffer structure so that we can play back the audio from the client.

**File Transfer Selected**

If the client decides they want to download a file we will wait for them to give us a filename and check to make sure that we actually have it. Once this has been checked and passes validation, we will start to read from the file using basic file I/O and send the data read to the socket for the client to read. Once we have sent the data, we will read more from the file and repeat this process until we reach end of file on the audio file.

## Multicast Set Up

### State Machine



### Pseudo Code

#### Client:

**Datagram Socket**

-Create a UDP Socket

**Multicast Address Initialized/Port Number Initialized**

* Initialize Multicast address from either user input or default value
* Port Number from either user input or default value

**Socket Bound**

* Create sockaddr struct
* Set address family in struct
* Set address (multicast address) in struct to listen from
* Set port in struct
* Call bind and check for error

**Multicast Group Joined**

* Set the multicast address as the destination address.
* Call set sock opt passing in this address and the correct parameters so that we are connected to the multicast sever. Make sure to error check so we are sure that we have joined or not.

**Waiting For Data On Socket**

* Simply call read on the socket and wait for data to come.
* Once data received process it.
* Note: this will be in its own thread so we can get more data and process/playback it at the same time.

**Process Data**

* Add data to a buffer
* Once buffer is full enough we will start to playback the audio.

#### Server:

**Datagram Socket**

* Create a UDP Socket

**Multicast Address Initialized/Port Number Initialized**

* Initialize Multicast address from either user input or default value
* Port Number from either user input or default value

**Multicast Interval Set & Set Time to Live**

* Get interval from user or if nothing is entered use default value.
* Get the TTL from the user, or set it as default value.
* Calls set sock opt to set the TTL (This shouldn’t be more than 2.) We really don’t need it but it is a good feature to have for scalability.

**Socket Bound**

* Create sockaddr struct
* Set address family in struct
* Set address (multicast address) in struct to listen from
* Set port in struct
* Call bind and check for error

**Multicast Group Joined**

- Set the multicast address as the destination address.

- Call set sock opt passing in this address and the correct parameters so that we are connected to the multicast sever. Make sure to error check so we are sure that we have joined or not.

**Loop back disabled**

* Set flag to false
* Use the flag in the set sock opt socket to disable loop back.
* This allows us to not get the messaged we are sending to the multicast group

**Destination Address Set to Multicast Addr**

* Define sockaddr in struct
* Set struct family
* Set struct address to multicast address
* Set struct port to port set earlier.
* This will let us call write and it will go to this address.

**Data Read From File**

* Open file in the playlist
* Read some data from the file

**Data Sent to Multicast Group**

* Send the data we read to the multicast group.
* Go back to reading more data.
* Repeat these steps forever. Replaying the playlist over and over.

## TCP File Transfer

### Flowchart



Pseudo Code

#### SERVER

**StartupSocket**

Create Socket

call Startup (WSAStartup)

load created socket (WASSocket)

Create socket options struct (SOCKADDR\_IN)

bind socket (bind)

**ListenForConntection**

call listen on socket (listen)

Create listening event (WSACreateEvent)

Create worker thread to handle Socket IO later on

**AcceptConnection**

loop

accept connection

set Accept event to false

**ListenForCommand**

Read from socket

if Microphone command is read, goto microphone mode

Otherwise, continue if FileIO command is read

**SendListOfValidAudioFiles**

Get internal list of Playable Audio Files

Write list to Socket

**WaitForResponse**

Read from socket

if data = valid name of a Playable audio file, continue

If not, return to SendListOfValidAudioFiles()

**OpenFile**

create a file pointer

Open the file indicated by the client data

**FileIO**

read data from file into buffer

if buffer is full or EOF is reached, continue

**SendPacket**

Write FileIO buffer to socket

if EOF is included, return to DisplayFileList()

otherwise, return to FileIO()

#### CLIENT

**StartupSocket**

Create Socket

call Startup (WSAStartup)

load created socket (WASSocket)

Create socket options struct (SOCKADDR\_IN)

**ConnectToServer**

Call connect on socket

**TransmitDataTransterRequest**

Get user input

if Microphone mode, send command for microphone mode and goto StartMicrophoneMode

if FileIO, send command for FileIOMode and continue

**DisplayFileList**

Read data from socket

display data to screen

**SendInputtedFileName**

Get user input

load input into buffer

write buffer to socket

**WaitForResponse**

read data from socket

put data into buffer

If invalid, return to DisplayFileList()

otherwise continue

**OpenFileToWrite**

open file to write

**WritePacketToFile**

write data from socket buffer to file

## UDP P2P Microphone

### Flowchart



### Pseudo Code

#### Client

**Client connects to server**

{

Connects to server

Display Server name and IP Address

}

**Mic Session Request**

{

Get Client Socket descriptor and other resources

Create an array packet

insert a voice chat request ID

Send off the packet to the server

Wait until it receives a response (blocked)

If it receives an acknowledgement

{

Call Mic Session created/initialized function

}

}

#### Server

**Server Waits for Client Connections**

{

Listen for client connections

new socket descriptor accepts the client

Display IP Address

}

**Process Client Request**

{

Get Server Socket descriptor and other resources

Waits until it receives a request from the client

If the request is a voice chat request ID

{

Echo back to client the request ID

Create UDP Socket()

Call Mic Session created/initialized function

}

}

#### Both

**Mic Session created/initialized (client/server)**

{

Create one thread that starts (monitoring for data from socket())

Create another thread that starts (monitoring for mic input())

}

**Monitor Mic Input (thread function) (Both Client/Server)**

{

if there is an input device

while true

{

Call WavInOpen function

store audio in buffer

call send audio data/packets function()

}

}

**Monitor packets for Data (thread function) (Both Client/Server)**

{

Initialize Socket Attributes

Bind passed in socket

while true

{

If we received something from the socket

store packets in buffer

The Buffering sound and playback completion callback routine will handle this

}

close socket descriptor if an error occurred

}

**Send Audio Data/Packets (Both Client/Server)**

{

Get Socket Descriptor and other resources

Send to the client/server

}

**Buffering Sound and Playback**

{

Pass in Sound Buffer

if we received something from the buffer and output device is detected

Playback audio buffer

clear buffer

}

**Create UDP Socket**

{

Create UDP Socket

}

**Bind Socket**

{

Bind UDP Socket

}

**Playback Audio function**

{

if there is an output device

Call WaveOutOpen function

}