February 16, 2015

Rhea Lauzon // Jeff Bayntun // Michael Chimick // Julian Brandrick

4O

Data Communications (Comp 4985)

Comm Audio

Contents

[Requirements 2](#_Toc413174756)

[Server: 2](#_Toc413174757)

[Client: 2](#_Toc413174758)

[Specifications: 3](#_Toc413174759)

[Multicasting 3](#_Toc413174760)

[Control Channel 3](#_Toc413174761)

[Data Channel (Peer-To-Peer) 3](#_Toc413174762)

[Voice Chat (Peer-To-Peer) 3](#_Toc413174763)

[Sending File 3](#_Toc413174764)

[State Flow Diagrams 4](#_Toc413174765)

[Server Side (High Level) 4](#_Toc413174766)

[Client Side 5](#_Toc413174767)

[High Level 5](#_Toc413174768)

[Client (Peer-To-Peer) 6](#_Toc413174769)

[Client (File Download) 7](#_Toc413174770)

[Message Protocol 7](#_Toc413174771)

[Messages: 7](#_Toc413174772)

[Pseudocode 9](#_Toc413174773)

[Server 9](#_Toc413174774)

[Client 11](#_Toc413174775)

[Client Multicast 11](#_Toc413174776)

[Client Peer-To-Peer 11](#_Toc413174777)

[Client File Transfer 12](#_Toc413174778)

[Client Microphone 12](#_Toc413174779)

[Helpers 12](#_Toc413174780)

[Unregister Multicast 12](#_Toc413174781)

# Requirements

* Create an audio streaming program
* Able to send sound data using UDP
* Must transfer data between two windows workstations and play the music
* Must have a Windows Interface
* Must be able to save and retrieve sound files provided
* The default sound file shall be in the .wav format
* Two way microphone support must work
  + Can between any two machines
* Multicasting capability
* Completion routines
* Server and client can be written as different programs

### Server:

* Able to transfer and play sound
* Sends the same data to every client at the same time (streaming) like radio
* Sends peer to peer music (for download)
* Can specify subnet address and port

### Client:

* Must be able to connect to a known remote server/workstation
* Able to transfer and play sound
* Client slightly behind server for buffering
* Able to download songs
* Can specify an IP and port

## Specifications:

### Multicasting

* Specified subnet address on both ends

## Control Channel

* Hard coded control channel port num
* Get the IP from fetching it from the multicast

### Data Channel (Peer-To-Peer)

* UDP data channel both ways
* Port via control channel + 1
* IP from multicast

### Voice Chat (Peer-To-Peer)

* Separate data channel (UDP) from data-channel

### Sending File

* Via secondary TCP channel

# State Flow Diagrams

## Server Side (High Level)



## Client Side

### High Level



### Client (Voice Chat)



### Client (Peer-To-Peer)



### Client (File Download)



# Message Protocol

There are several messages that are sent between the client and server via the control channel once it is established. The control channel is established after the multicast is established. The following is a description of all messages that will be sent between the two and their description.

Message Format: MESSAGE\_TYPE~DATA`

The message type and data are delimited by a ‘~’ for separation and is not part of either. The end of the message is delaminated by a ‘`’ which is also not part of the data.

## Messages:

Color Key:

|  |  |
| --- | --- |
|  | Server->client |
|  | Client->Server |
|  | Either side |

|  |  |  |
| --- | --- | --- |
| MESSAGE\_TYPE | DATA | Description |
| END\_CONNECTION | No data | Message from client to server or server to client specifying they will be disconnecting and the control channel will be removed |
| MIC\_CONNECTION | No data | Message from client to server or another client for mic connection |
| SONG\_REQUEST | Name of song to be played | Message from client specifying the peer-to-peer song that they would like to listen. This should trigger a UDP connection between the client and server for the transfer of the requested song. |
| SAVE\_SONG | Name of song to be saved | Sent by client to the server to request a song to be saved. |
| LIBRARY\_INFO | Name of each song that is available from the server. Each song should be separated by a ‘|’, and the artist and song name are separated by a ‘^’  EX: Imagine^John Lennon|Let It Be^The Beatles | Message from server to client upon multicast connection. The server sends the name and artist of all songs available to the client. This data is then used to display a choice of songs for peer-to-peer listening. |
| NOW\_PLAYING | Name of the song playing current with the corresponding information separated by a ‘^’  Format: name^artist^album^length  EX: Imagine^John Lennon^Imagine^3:01 | Message sent from server to client for the song current being played. |
| END\_SONG | Empty | Server to clients indicating the current song is done |
| CURRENT\_LISTENERS | List of IPs of currently listening devices  Format: IP|IP|IP | Server to client to specify all currently listening clients and its own IP for microphone capabilities. |

# Pseudocode

## Server

1. Initialize Server function
2. {
3. Load list of song titles and locations
4. Initialize Multicast thread // details below
6. create Accept thread
7. }
8. CreateListen function
9. {
10. create session semaphore, initialize to 1
11. create a socket and Listen on SERVER TCP LISTEN PORT
12. while true
13. when a new connection arrives, create new Session to handle connection
14. allocate new session structure
15. }
16. CreateSession function
17. {
18. wait on session semaphore
19. add allocated session structure to sessionMap
20. signal on session semaphore
21. create control thread
22. }
23. ControlThread function
24. {
25. establish session from input parameter
26. create finishedTransfer semaphore, initialize to 0
27. create userChanged semaphore, initialize to 0
29. send song list
30. send name/artist/album of currently playing song
31. update clients with new user connection
33. make TCP Receive call with controlRoutine as call-back
34. while true
35. enter alertable state and wait for the previous two semaphores
36. if new user connected
37. send new user list
38. if unicast/TCP song finished
39. send END\_SONG message
40. if error occurs
41. clean up session
42. }
43. Control Routine
44. {
45. if received data
46. if unicast requested
47. wait on session semaphore
48. establish session from socket used for transfer
49. signal on session semaphore
50. give session structure the requested filename
51. create unicast thread with session
52. if TCP file requested
53. create TCP file thread
54. if close
55. start cleanup
56. if improper message
57. start cleanup
58. if sending
59. if still data to send, send again
60. else do TCP Receive call with callback to ControlRoutine
61. }
62. Unicast Thread
63. {
64. create UDP socket
65. pass socket to waitingClients queue
66. start sending the song through to the client
67. when sending is finished, signal on finishedTransfer semaphore
69. terminate Unicast thread
70. }
71. TCPFile function
72. {
73. get session based on socket
74. initialize TCP socket
75. send file stream
76. when sending is finished, signal on finishedTransfer semaphore
78. terminate TCPFile thread
79. }
80. CleanupSession function
81. {
82. wait on session semaphore
83. close finishedTransfer semaphore
84. close userChanged semaphore
85. signal on session semaphore
87. send new user list
88. exit session control thread
89. }

## 

1. Multicast Thread function
2. {
3. initialize multicast socket
4. create multicast session
5. set time to live
6. disable loopback
8. initialize multicast song list
9. start playing song over multicast socket
10. }
11. Multicast Thread function
12. {
13. Start Multicast
15. while server not in shut down state
16. run Music Multicast
18. Stop Multicast
19. }
20. Start Multicast function
21. {
22. initialize resources
23. start multicast session
24. }
25. Music Multicast function
26. {
27. choose random song from song folder
29. set currently playing song to picked song
31. load song
33. set up callback for UDP music sending to Multicast Routine
34. }
35. Multicast Routine function
36. {
37. send audio file until it has all been sent
38. }
39. Stop Multicast function
40. {
41. communicate end of session to all clients
43. end multicast session
45. clean up used resources
46. }

## Client

1. Idle function
2. {
3. Initialize GUI
4. Call join multicast function
5. }
6. Join Multicast function
7. {
8. Add membership to multicast session
9. Start recieving thread
10. Call playing function
11. Make sure other thread is dead
12. Kill thread
13. }
14. /\*
15. The following functions require these initial conditions:
16. binary semaphore mRecv = 1
17. binary semaphore mPlay = 0
18. semaphore mBuffer = size of multicastBuffer
19. \*/
20. Multicast Recv function
21. {
22. Open udp listening socket
23. while die signal not given
24. {
25. wait for data
26. if the data is song data
27. wait semaphore mBuffer
28. wait semaphore mRecv
29. place data into multicastBuffer
30. signal semaphore mRecv
31. signal semaphore mPlay
32. if the data is library information
33. update songs list and GUI elements
34. if the data is client information
35. update list of clients and update GUI
36. if the data is now playing
37. update now playing GUI
38. }
39. cleanup
40. kill thread
41. }
42. Multicast Play function
43. {
44. open local speakers
45. while die signal not given
46. {
47. wait semaphore mPlay
48. wait semaphore mRecv
49. get data from multicastBuffer
50. signal semaphore mRecv
51. signal semaphore mBuffer
52. play data on speakers
53. }
54. cleanup
55. // don't kill, this is the parent thread
56. }
57. display songlist function
58. {
59. for each song in the list of songs available from the server
60. {
61. add the song and artist name to a clickable list item on GUI
62. update GUI
63. }
64. }
65. request song stream function
66. {
67. get the song name from the GUI item clicked
68. generate a control message for song request with the message data set to
    1. the song name
69. open the UDP channel for receiving data
70. send the control message on the TCP control channel to the server
71. call the unregister from multicast function
72. call the receive song information function
73. }
74. receive song information function
75. {
76. get the song name, artist, and album
77. set the GUI track player with the data
78. call the play single song function
79. }
80. play single song function
81. {
82. while the song has not finished
83. receive song bytes from the UDP channel and store them in the buffer
84. play the buffer
85. close the UDP channel with the server
86. register for multicast
87. }
88. request song download function
89. {
90. get the song name from the GUI item selected
91. generate a control message for SAVE\_SONG with the message data set to
    1. the song name
92. create file transfer thread
93. send the control message on the TCP control channel to the server
94. }
95. File transfer thread
96. {
97. while true
98. Create a socket and listen for TCP connections
99. when a new connection arrives, validate
100. if valid, create new connection
101. begin receiving file data until all data is received
102. }
103. // Should be started as a thread
104. Start Voice Chat function
105. {
106. Start sending thread
107. Start recieving thread
108. Start recording thread
109. Call sound playing function
110. Make sure other threads are dead
111. Kill thread
112. }
113. /\*
114. The following functions require these initial conditions:
115. binary semaphore OutRecord = 1
116. binary semaphore OutSend = 0
117. semaphore OutBuffer = size of voiceOutBuffer
118. binary semaphore InRecv = 1
119. binary semaphore InPlay = 0
120. semaphore InBuffer = size of voiceInBuffer
121. \*/
122. // All the following functions would be in a separate thread
123. Voice Record function
124. {
125. open local microphone
126. while die signal not given
127. {
128. record data
129. wait semaphore OutBuffer
130. wait semaphore OutRecord
131. place data into voiceOutBuffer
132. signal semaphore OutRecord
133. signal semaphore OutSend
134. }
135. cleanup
136. kill thread
137. }
138. Voice Send function
139. {
140. open udp sending socket
141. while die signal not given
142. {
143. wait semaphore OutSend
144. wait semaphore OutRecord
145. get data from voiceOutBuffer
146. signal semaphore OutRecord
147. signal semaphore OutBuffer
148. send data on socket
149. }
150. cleanup
151. kill thread
152. }
153. Voice Recv function
154. {
155. Open udp listening socket
156. while die signal not given
157. {
158. wait for data
159. wait semaphore InBuffer
160. wait semaphore InRecv
161. place data into voiceInBuffer
162. signal semaphore InRecv
163. signal semaphore InPlay
164. }
165. cleanup
166. kill thread
167. }
168. Voice Play function
169. {
170. open local speakers
171. while die signal not given
172. {
173. wait semaphore InPlay
174. wait semaphore InRecv
175. get data from voiceInBuffer
176. signal semaphore InRecv
177. signal semaphore InBuffer
178. play data on speakers
179. }
180. cleanup
181. // don't kill, this is the parent thread
182. }