Project report QtheMusic

Denhi Huynh
dat11dhu@student.lu.se

Fredrik Folkesson
dat11ffo@student.lu.se

Johan Nyholm
dat11jny@student.lu.se

Shan Senanayake
dat11sse@student.lu.se

May 21, 2015

1 Background

In most festive events or parties today, there is one computer which plays the music at the event. In order to add a song to the music queue one has to walk to this main computer. If many people would like to queue songs at once, there can be a build up by this computer since everyone wants their music to be played. Our team would like to solve this problem. Therefore, we have created an android application and a java desktop application in order to queue music to a specific computer from any device at a given event.

2 Requirement specification

In the beginning we discussed which requirements our team wanted for our system. The discussion resulted in the following requirements for a first release:

- The host should support multiple simultaneous clients.
- The queue should be updated throughout all the clients when a change has occurred.
- The program should use the network.
- The program should be written in mostly Java
- The host should be able to edit the music queue.
- The clients should only be able to view and append the queue.
- The project should consists of three parts, a host, client and central server.

If more time was available, we decided to attempt to implement the following features:

- Make an Android client and host.
- Set up a real central server for example on a raspberry pi.

3 Model

The project consist of mainly three parts the central server, the user host and the user client.

3.1 Central Server

The purpose of the central server is to set up connections between the user host and the user client. The original idea of the central server was to give out information about the available host to the client when a connection to the central server is made. The information could contain data about location, name etc. However, it currently only works as a glorified DNS server.

3.1.1 Connection between Central server and client

The central server acts as a hub between host and clients. When a client makes a connection to the server, it gives out which available host ids that exists. Then the client sends a host id to the server which then responds with the IP-address to the host. Then the client-server connection dies.

3.1.2 Connection between Central server and host

When a host connects to the server, the hosts sends it's IP-address to the server and the server responds with a id which defines the host id. The server will only send this id if there exists any available ids, this is so that the server will not be overloaded by the hosts. This connection persists until the host is ready to die. If the connection dies then the server resets the id for that host and makes it available again.

3.1.3 Classes

The central server consists of several thread-classes that represent the connection to and from the clients and hosts. All the classes that represent the central server will be explained below.

`UserHostListener` This thread represents the `ServerSocket` which listens to new hosts wanting to connect to the server. When a host connects a new thread-class `HostAddressPutter`.

`HostAddressPutter` This thread sets up a host id for the connecting host if such a id exists. Then it keeps listening to the host for it to die and then resets the id that was given.

`UserClientListener` This thread represents the `ServerSocket` which listens to new clients wanting to connect to the server. When a host connects a new thread-class `HostAddressFetcher`.

`HostAddressFetcher` This thread gives out all the host information to the client and then waits for a response then it gives out the hosts IP-address information. After this the thread dies.

`ServerMonitor` This class represents the monitor of the central server and handles all the sensitive information that has to be thread safe. Such as ids and IP-addresses.

3.2 User Host

The User Host is essentially the music player. It has the option to stop, skip, play and pause the queued songs. The original idea was to let the user select a folder from which the User host would play music. Though currently all songs reside in a specific folder.

3.2.1 Request an ID

The first thing done by the User Host is to register at the Central Server. The Central Server then provides the User Host with an ID with which the connection details of the User Host is linked. User Clients can then look up the connection details of the User Host using the given ID. The host will then keep a connection with the Central Server for the rest of the hosting session.

3.2.2 Process incoming connections

The User Host listens to incoming connections from User Clients. The host currently supports a maximum of 10 clients connected at any time. Each time a connection to a User Client is requested and established, the User Host sends the list of available tracks to it. The content of current queue is then sequentially sent.

3.2.3 Listen to requests

All connected User Clients may send queue requests at any given time. The User Host listens to these requests and in turn updates the queue accordingly.

3.2.4 Updating User Clients

Each time an event occurs where the content of the queue is changed, all connected clients gets notified about the event. Making sure that the queue is always synced.

3.2.5 Classes

The User Host consists of several thread-classes that represent the connection to and from the User Clients as well as a connection to the Central Server.

`UserHost` This serves as the main thread. It starts all threads and initiates the User Host service.

`MusicPlayerThread` This thread fetches the latest song from the `HostMusicQueue`, lets the `MusicPlaybackThread` play it and waits for the queue to receive status updates from the queue.

`MusicPlaybackThread` This thread plays an mp3 file. It informs the `HostMusicQueue` when the song is finished.

`HostMonitor` All connections to `User Client`s connected to the `User Host` are stored in this monitor. It contains a lost of messages to be sent to clients as well as methods for communicating with clients. Each time changes are made, messages are added to a list of outgoing messages, containing information about the event with the connected clients as recipients. These messages can then be sent calling a sent from within the monitor.

`QueueActionMessage` A sort of container of information regarding an action or event that has changed the queue. In this class is also a list of `User Client`s which needs to be informed about this event.

`HostIDFetcherThread` The HostIDFetcherThread registers the host at the `Central Server` and receives the ID assigned to it. This thread establishes and holds a connection to the `Central Server` until the host withes to quit. In case the connection is somehow lost, the `HostMonitor` is informed about the event.

`IncomingClientListenerThread` This thread starts a `HostToClientWriterThread` and then listens to connections from clients. Every time a client successfully connects, it is registered in the `HostMonitor` and a `HostToClientReaderThread` is started, which in turn listens to the clients requests.

`HostToClientWriterThread` This thread serves as the mailman (or woman). It sends all messages residing in the `HostMonitor` to the `User Client`s listed in each message.

`HostToClientReaderThread` Each connected `User Client` has a designated thread of this sort. This thread listens to incoming requests from the designated client. Each request is then added to the `HostMonitor`. In case the client disconnects, the monitor is also notified.

3.3 User Client

The User client is the main application that is used to queue up songs. The client gets the available tracks from the host when the connection is first established. The user can then add any song from the available songs to the user hosts queue. A change on the user hosts queue is updated on every client connected to that host

3.3.1 Connect to a user host

The client first requests a host id from the user. Then the client connects to the central server and requests the IP address for that host id. A connection between the user host and the client is then made.

3.3.2 Queue a song

To queue a song the client sends a message consisting of the string "Q number" (where number is the track id the client wants to queue) to the user host

3.3.3 Receive updates about the queue

The client listen on the host for updates regarding the queue (adding of new songs, starting of songs, stopping of songs)

3.3.4 Classes

The client consists of two threads. One thread listens to input from the user and sends messages to the user host. The other thread listens to input from the user host and updates the queue representation on the client

`UserClient` The user client retrives the host address and starts the hostlistener thread. After that it listen to input from the user and sends queue messages to the user host.

`HostAddressRetriever` This class does the actual retrieving of the hosts ip address from the central server using a Host ID number.

`ClientFromHostReaderThread` This class listens to input from the user host and updates the queue on the client side.

`ClientMusicQueue` This is a monitor, which represents the client side of the music queue. This class has method for updating and reading the current state of the queue.

4 User manual

4.1 Android Application

Setup:

- Make sure that all users (hosts and clients) are connected to the same wifi.
- Start the app
- Choose if you want to be a party host or party client.

If the steps described in setup are followed, then the figure 1 should be displayed.

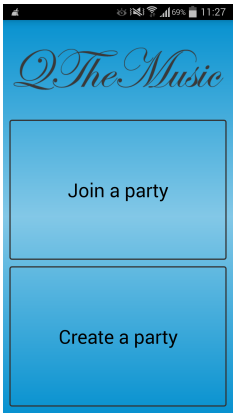

Figure 1: The home screen.

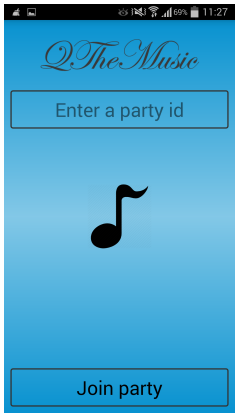

Figure 2: The type host id view.

Join a party:
When "join a party" option is chosen, the user is presented with a screen where the user can input the party host id, see figure 2. If the id exists and the users are using the same network, the client view is shown, see figure 3.

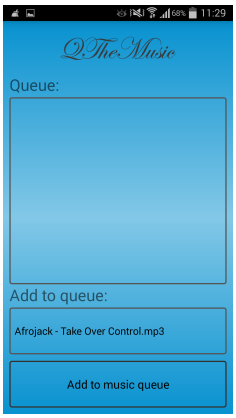

Figure 3: The user client view.

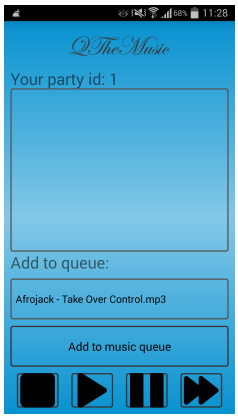

Figure 4: The host view.

Create a party:
When create a party is chosen, the user is presented with a party host view. From this view the user can add songs and also control the music player, see figure 4.

4.2 Desktop Application

The host and clients also exists as a desktop java application if you want to run it on your computer and not your phone. The hosts and clients on the android application and the desktop version is interchangeable, which means one can have the host on a computer and the clients on phones.

5 Evaluation

5.1 Improvements

Despite the fact that we are satisfied with whats been accomplished, the system is still far from finished. In a publicly released version the central server would have been implemented to handle the traffic, instead of having it as a DNS, which it currently is. The Music player aspect of the could be better implemented, for example the play and pause button could be a single button which changes the option instead of having two separate buttons.

Most persons today also use other sources of music than a folder containing mp3 files. A future implementation would connect this application to other music services such as Spotify or Tidal.

As of now, the folder path is hard coded both on the android app and the desktop application. In a future version, this would be dynamically set by a user.

5.2 Thoughts about the project

This project has been a great way of applying the learned knowledge from the network programming course. The work load has been appropriate. However, it is very easy to want to do more than what time allows.

6 Program code

The eclipse project can be found here (click this text).

The Android studio project can be found here (click this text).