Kompose

A Distributed Playlist for Android

Mark Arnold, Dino Bollinger, Tobias Brodmann
15-917-701, 14-923-676, 15-934-565
arnomark@student.ethz.ch, bdino@student.ethz.ch,
brotobia@student.ethz.ch

Lino Lendi, Florian Moser, Lukas Tobler
11-714-383, 15-930-704, 14-942-007
llendi@student.ethz.ch, moserfl@studen.ethz.ch,
lutobler@student.ethz.ch

ABSTRACT

Kompose is an Android application that aims to allow multiple independent participants to form local sessions through which they share a music playlist and request for songs to be played at social events. The songs will be downloaded from Youtube, and each participant will be able to share, synchronize and vote on what song they want to be played next, in a fully distributed fashion. The host will hereby act as the playback device, from which the songs will be played.

1. INTRODUCTION

At social events, such as clubs or parties, we often face the problem of deciding which person should be in charge of the music being played. From our experience, many people often disagree with a particular DJs taste in music, and are subsequently dissatisfied with the chosen playlist. And even if the DJ only takes suggestions from the participants themselves, there may often be fringe tastes that are only enjoyed by a small minority. The DJ may also be overwhelmed by requests, and would benefit greatly from a simple way for people to keep track of what has already been requested.

Kompose solves this problem by creating a distributed playlist that anyone can add songs to. One device will start a session (the *host*, usually connected to a stereo), and other devices on the local network can join in and start adding songs to the queue. Apart from choosing the name for the party and being responsible for the playback, in terms of voting, the host will have the same rights as the other clients in the system. Clients can *downvote* a specific song, and once a majority of clients downvoted a song, it will be removed from the queue. Past sessions will be kept, so the users can review playlists at a later date. The primary source of music will be YouTube. Clients send their song request in the form of a youtube URL to the host.

The challenges in this project are as follows:

- Automatically announcing and finding other *Kompose* services on the network.
- Synchronizing the playlist state to all devices.
- Keeping track of all clients that are still alive to allow majority based voting.
- Adding new songs to the playlist, and voting on skipping songs, which should be consistent and fair for all clients.
- Fault tolerance and error handling in case the host or a client in the network crashes.
- Finally, a protocol that supports all these features must be devised.

2. SYSTEM OVERVIEW

2.1 Client/Server architecture

Each device running the app can choose to become host, at which point it will act as the server of a session. All communication is done over these hosts, i.e. there is no peer-to-peer communication between other clients. To support this, we define a custom application-level communication protocol which provides at least the following functionality:

- Client registration and deregistration messages.
- Song requests
- ”Vote-Skip” messages, in the form of downvotes.
- ”Keep-Alive” messages, which are periodically sent to make sure participants are still active.
- Playlist state, which includes all songs in the play queue, as well as the received downvotes for each of the songs
- Error responses for invalid requests.

The messages of this protocol will be transported over TCP in the form of JSON.

Both client and server listen on the network for messages, the host for requests from the clients, and the clients for state updates from the host. To avoid manual IP/port configuration, *Network Service Discovery* [1] is used. This gives the clients the hosts' IP and port, to which they can send a *register* message containing the IP and port they will be listening on for state updates.

Once a client is connected to the host, he can start sending song request and vote messages. Song requests will cause the corresponding song to be added to the playlist, and *Vote-Skip* messages help make the participants reach consensus over which songs they want to play. Once a song receives skip requests from a majority of the clients, the song will be removed from the playlist.

To add some resilience against clients dropping off the network, clients periodically send a keep-alive message to their host every few seconds, and vice-versa. If a client becomes unresponsive, he will be dropped from the party and the maximum number of votes necessary to remove a song from the playlist will decrease accordingly. This serves to ensure that the voting system does not become unresponsive.

In the case where the host crashes or become otherwise unreachable, the party will be disolved gracefully. If time allows, we may also explore options to dynamically switch the host to one of the existing party members while reusing the existing playlist.

2.2 User interface

The *Kompose* app will consist of at least 6 different activities in total, each serving a unique purpose, as well as several dialogue messages which will offer further options to the user.

Figure 1 shows a mockup of this user interface design. The start screen (top center) allows the user to either join a pre-existing party, or create an entirely new one. When creating a new party, the host chooses a new name which will be visible for all clients in the LAN. It then proceeds to setup the playlist and send out service discovery messages. Here he will continuously listen for new connections, song requests and votes.

When choosing to join a party, the clients will be sent to a screen that displays a list of all currently available parties in the LAN. If an announcement from a host is received, the party name shows up in this list and can be joined. Once the choice is made, the client is sent to the main playlist interface.

In addition to these two options, there is also a third which will send the user to the history of all playlists of previous sessions. This mainly serves to help the user look up past songs that he may have forgotten the name of. This feature is non-critical for the primary function of the app, but it is nonetheless a welcome service for the user.

The main playlist interface will display a list of all items currently in the play queue. This queue is displayed in the order in which songs will be played by the host, from first to last in line. Each list element will display at least the title of the file, as well as the URL from where the audio has been retrieved. Finally, each list item includes a designated downvote button, which serves to have unpopular songs be removed from the queue. As soon as a majority of the clients deem a song bad, it will no longer appear in the queue.

The host will additionally have a play/pause button in the action bar.

2.3 YouTube file downloads

Songs are fetched from YouTube by the host only, using a library [2] to extract the URL for the audio file. Songs in the play queue are pre-fetched one song ahead of time to guarantee continuous playback. The number of items that are cached can be configured in the settings. This is of course handled asynchronously in the background. The playlist is stored as strings of text, which is also what will be transported to the clients.

3. REQUIREMENTS

Discovery service.

We will use the Android Network Service Discovery API [1] as provided by Android itself to enable the client-server discovery in the network

YouTube Downloads.

We will enable users to put videos from youtube inside the play queue. To fetch the audio URLs from those videos we will rely on a library called android-youtubeExtractor [2]

4. WORK PACKAGES

To work efficiently, and enable group members to work in their area of expertise we will divide the group coarsely into two teams, which we will henceforth refer to as the "technical team" and the "design team" respectively. With this in mind, our work packages are split by technical and design aspects, but the designers and the developers will stay in close contact throughout the whole process.

WP1: Protocol Specification.

The technical group will first define a protocol to support all required functionalities. A significant amount of time will be spent on planning, because a well-defined protocol will aid in creating a cleaner, more goal-oriented implementation that will be easier to update further down the line. The target of this work package is the definition of a small but extensible protocol.

WP2: Client/Server Implementation.

This work package includes the implementation of the defined protocol in *WP1* with all functionality as defined in this document by the technical team. The target of this work package is a fully functional prototype.

WP3: User Interface Design.

Parallel to WP1 and WP2 the design team will work on the look and feel of the interface. Besides the design, the group will focus on usability aspects, and will be in close contact to the technical team to help with proper backend integration. The target of the work package is a consistent design of the various pages which fits to its target user group.

WP4: UI Implementation.

After finishing the design it will be implemented using the prototype from *WP3*. Target of this work package is the nearly finished application.

WP5: Testing & Polish.

For this last work package the design and technical team will work together on finalizing the functions and design of the application. Inputs from people from outside the projects will also be evaluated. Target of this work package is to polish the design, and confirm that all functionality works as intended and as advertised in the UI.

5. MILESTONES

5.1 Milestone technical

The technical milestone will include both *WP1* and *WP2* and will be done by our technical team, consisting of the members lutobler, brotobia and bdino. They have already started with their work packages, and will finish the milestone by 25.11.17 at latest.

5.2 Milestone design

The design milestone will include both *WP3* and *WP4* and will be completed by our design team, consisting of moserfl, llendi and arnomark. While *WP3* is already in progress, *WP4* will start after *Milestone technical* is completed. Both work packages will be done by 02.12.17

5.3 Milestone testing

After the completion of the first two milestones, all team members will come together to test and polish the application as described in *WP5*. The testing phase will finish on 14.12.17 at the latest. Shortly thereafter the application and all additional material will be submitted. This will respect all of the fixed deadlines:

- **15.12.2017:** Presentation/Logo deadline
- **17.12.2017:** Code deadline
- **18.12.2017:** Presentation and demo session

6. REFERENCES

[1] Android API: Network Service Discovery. `https://developer.android.com/training/connect-devices-wirelessly/nsd.html`. Accessed on 13 Nov 2017.

[2] Android based YouTube URL extractor. `https://github.com/HaarigerHarald/android-youtubeExtractor`. Accessed on 13 Nov 2017.

[3] Balsamiq UI mockup desgin tool. `https://balsamiq.com/`. Accessed on 13 Nov 2017.

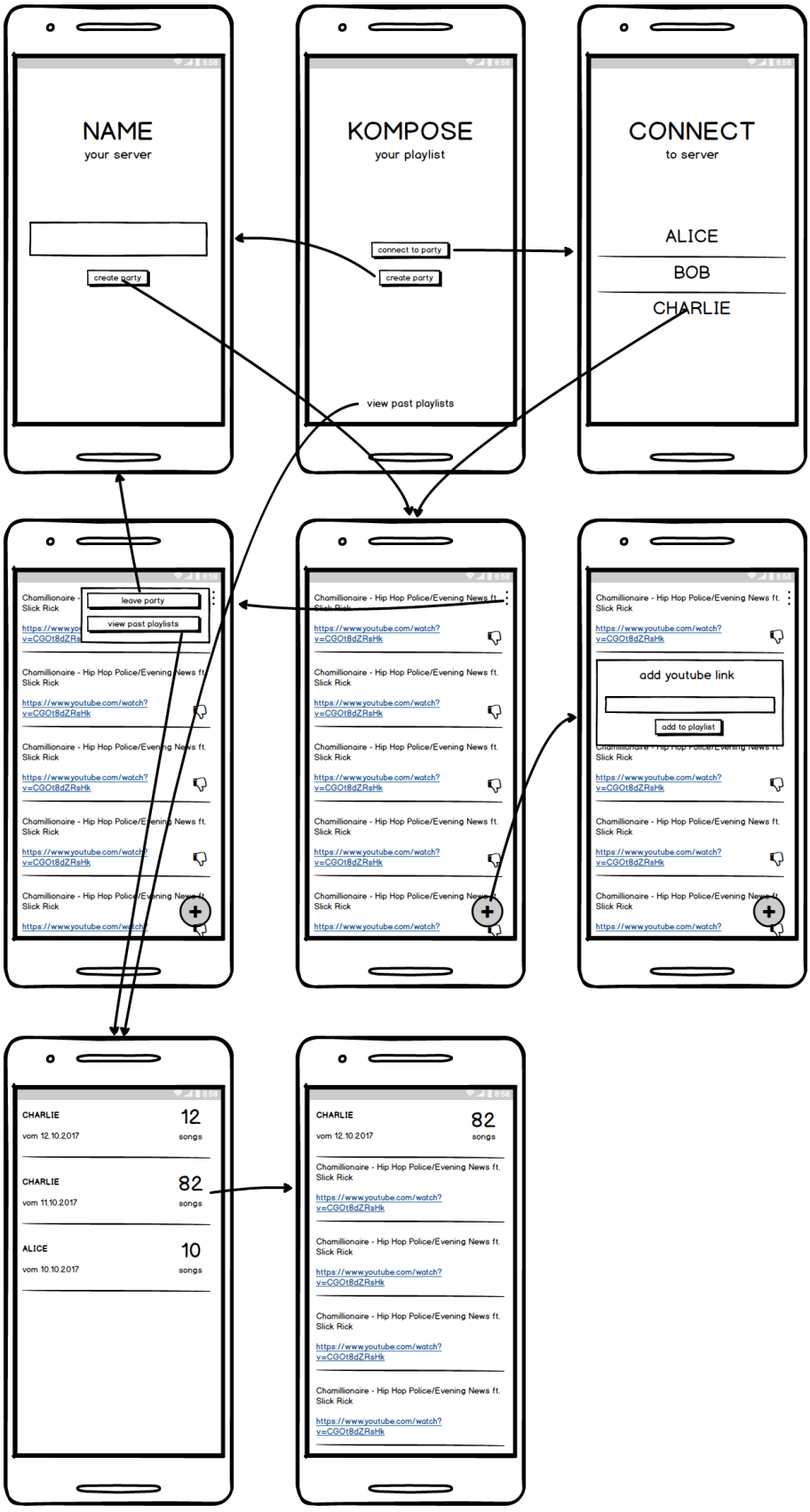

Figure 1: User interface mockup (created with [3])