WhatsPlaying Music Deployment

Projektarbeit

Böffel, Matthias
Nebel, Markus
Sauer, Janina

22. Februar 2016

Betreuer: Prof. Dr. Manh Tien Tran
Veranstaltung: Mobile Anwendungen mit Android
Studiengang: Informatik (Master)

Inhaltsverzeichnis

1 Projektbeschreibung

Im Rahmen der Veranstaltung *Mobile Anwendungen mit Android*, betreut von Prof. Dr. Manh Tien Tran, wird eine Android App entwickelt, die zur zentralen Musikwiedergabe von auf anderen Smartphones verteilten Musikstücken dient. Auf einem zentralen Smartphone werden die Musik-Titel der verbunden Smartphones in einer Playlist gesammelt und anschließend in einer definierten Reihenfolge am zentralen Smartphone abgespielt.

Ziel ist die Entwicklung zweier Apps, die das Server-Client-Konzept umsetzen. Die Server-App wird dabei auf dem zentralen Smartphone ausgeführt. Die Client-App verbindet sich über ein Übertragungsstandard (z.B. Bluetooth oder WLAN/TCP) mit dem Server und teilt diesem seine lokale Client-Playlist mit, die der Anwender zuvor aus seinem lokalen Smartphone-Speicher zusammengestellt hat.

Der Server verfügt über einen Algorithmus, der die lokalen Playlisten aller Clients in einer gewissen Reihenfolge sortiert und dadurch eine globale Playlist erzeugt. Danach werden die ersten n Musikstücke aus der globalen Playlist bei den jeweiligen Clients angefordert und versucht immer die nachfolgenden n Musikstücke in einem Puffer vorzuhalten. Letzteres soll bei Verbindungsproblemen eine unterbrechungsfreie Wiedergabe ermöglichen.

2 Motivation

Musik-Streaming-Dienste wie Spotify oder Amazon Prime sind mittlerweile, neben dem persönlichen Musikkonsum, zu beliebten Unterhaltungsplattformen bei der Veranstaltung privater Feiern herangewachsen. Ihr breites Angebot kann nahezug jedes musikalische Genre abdecken und ermöglicht es dem Gastgeber so die meisten Party-Gäste musikalisch bei Laune zu halten. Das Bedienkonzept dieser Dienste verlangt jedoch, dass die abgespielten Musikstücke entweder zuvor von einer Person in eine Playlist einsortiert wurden, oder ständig verschiedene Personen die nächsten Musikstücke manuell am Abspielgerät suchen und auswählen müssen.

Als Beispielszenario diene eine private Feier. Es ist heutzutage üblich, dass an der Musikanlage ein Laptop oder PC zur bequemen Gestaltung einer Playliste angeschlossen ist. Medium der Wahl ist oft eine MP3-Sammlung, die von Gästen mittels USB-Stick spontan ergänzt wird. Dabei entsteht ein gewisser Verwaltungsaufwand und die Bedienung des Abspielgeräts steht im Mittelpunkt jeder Anpassung der Playliste. Interessenkonflikte bei der Gestaltung der Playliste sind nicht ausgeschlossen und bei der Bedienung besteht durchaus auch die Gefahr, dass ein Getränk auf der Hardware ausläuft. Außerdem muss der Zugang zum PC gewährleistet sein.

Ein Smartphone als Server, angeschlossen an der Musikanlage und bedienbar durch andere Smartphones als Clients, löst diese Probleme. Dabei entfällt auch die manuelle Übertragung der Musik, weil diese sich bereits bei den Gästen auf den Client-Smartphones befindet. Das Server-Smartphone kann ggf. versteckt und gesperrt angeschlossen sein und übernimmt dann die Rolle der Verteilung von Abspielzeit durch festgelegte Strategien. Gäste können dabei jederzeit auf ihrer Client-App die globale Playliste mitverfolgen und werden optional benachrichtigt, wann das nächste eigene Musikstück abgespielt werden wird.

3 Techniken

3.1 ShowCaseView

Zur Gestaltung eines Einführungstutorials wurde (sowohl auf Server- als auch auf Clientseite) die Bibliothek „ShowCaseView“ verwendet. Mit Hilfe dieser Library können einzelne Elemente auf der Bildschirmseite hervorgehoben und mittels textueller Beschreibung zusätzlich erläutert werden. Abbildung 9 im Abschnitt 5 zeigt eine Beispielseite des umgesetzten Tutorials auf Clientseite.

Die Library wurde angepasst und mit kompiliert, um einen Bug zu beheben, der u.a. in Verbindung mit LG-Geräten auftrat, durch den die Breite von Views negativ interpretiert werden konnte, was die Darstellung beeinflusste.

3.2 Android Hintergrund-Prozesse

Um die Kommunikation zwischen Server und Client kontinuierlich aufrecht zu erhalten, sowie auf Serverseite den Mediaplayer fortwährend wiedergeben zu lassen, ist es notwendig die Logik in einen Hintergrundprozess zu verlagern, der unabhängig vom Lebenszyklus der Activity ausgeführt wird. Abbildung 1 zeigt diesen Aufbau schematisch.

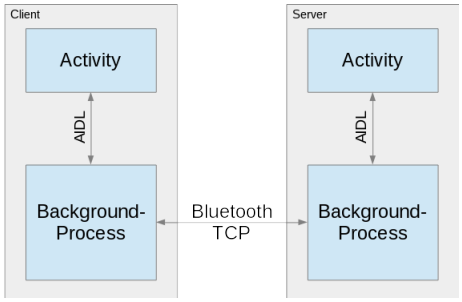

Abbildung 1: Kommunikation zwischen Activity, Hintergrundprozess und Server-Client

3.3 AIDL

Um die Interprozesskommunikation zwischen Activity und Hintergrundprozess möglichst einfach zu gestalten, wurde auf die *Android Interface Description Language* (kurz AIDL) zurück gegriffen. Mit AIDL wird die Komplexität deutlich reduziert und die Verwendung des Android-Binders vollständig vom Entwickler verborgen.

3.4 Übertragungstechniken

Durch die Schnittstellen-basierte Programmarchitektur ist es möglich die App um beliebige Übertragungstechniken zu erweitern. Implementiert wurden bereits Bluetooth und TCP. Das UML Diagramm in Abbildung 2 zeigt wie die Schnittstellen implementiert wurden und verdeutlicht die Erweiterbarkeit.

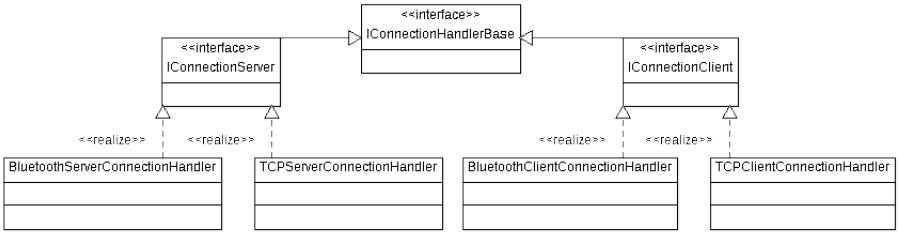

Abbildung 2: UML Klassendiagramm der Interface-Hierarchie der Konnektivität

3.4.1 Bluetooth

Da dank Bluetooth eine autarke Infrastruktur ohne notwendige zusätzliche Hardware verfügbar ist, wurden die Apps zunächst auf Basis von Bluetooth umgesetzt. Android Smartphones bieten zwar die Möglichkeit als WLAN-Access-Point aufzutreten, jedoch wird hierbei der Internet-Traffic der verbundenen Clients über den mobilen Datentarif des Server-Smartphones geroutet, was zu hohen Kosten führen kann.

Aufgrund der Instabilität und der geringen Übertragungsrate wurde zusätzlich zu Bluetooth im späteren Verlauf der Entwicklung jedoch TCP hinzugefügt, auch wenn hierfür unter Umständen zusätzliche Hardware notwendig ist.

3.4.2 TCP

Die Verwendung von TCP zur Kommunikation zwischen Server und Clients macht die Verwendung der App für den Anwender intuitiver, da außer der Anmeldung in einem zumeist PSK-gesicherten WLAN-Netz keine weitere Aktionen notwendig sind. Die Client-App ermittelt die Server anhand eines Broadcast-Pakets (über UDP), auf das jeder im Netz verfügbare Server antwortet.

Durch die üblicherweise deutlich höhere WLAN Bandbreite im Vergleich zu Bluetooth, wird die Übertragung der Musikdateien merklich schneller durchgeführt.

3.5 Buffering

Die App nutzt kein Audio-Streaming, sondern fordert jede Musikdatei kurz vor seiner Wiedergabe beim jeweiligen Client an. Die Dateien werden dann vollständig zum Server übertragen und dort temporär für die Dauer der Wiedergabe im Speicher gehalten. Dieser Mechanismus soll durch Verbindungsprobleme ausgelöste Wiedergabe-Probleme unterbinden. Der Server fordert eine konfigurierbare Menge an Musikdateien im Voraus an. Zusätzlich wird auf diese Weise rechtlichen Problemen im Bezug auf unerlaubte Vervielfältigung der Musikstücke Rücksicht genommen.

3.6 Lokalisierung

Das Android SDK bietet standarisierte Möglichkeiten zur Lokalisierung von Apps (Unterstützung mehrerer Sprachen) durch die Auslagerung der sprachrelavanten Strings in separate XML-Dateien. Von Beginn an werden die beiden Sprachen Englisch und Deutsch unterstützt. Die Spracheinstellung richtet sich nach der auf dem Android-Gerät eingestellten System-Sprache.

3.7 Android Wear

Eine Android-fähige Smartwatch zeigt bei bestehender Bluetooth-Verbindung auf Clientseite den aktuell abgespielten Musiktitel an (Abbildung 3). Auch wird über eine Notification durch Vibration darauf aufmerksam gemacht, dass in Kürze der nächste Musiktitel aus der lokalen Playliste abgespielt wird.

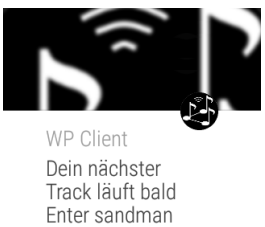

Abbildung 3: Screenshot Android Wear Notification

4 Software-Architektur

4.1 Server-Service

Der Hintergrund-Prozess der Server-App erzeugt zunächst, je nach genutztem Übertragungsstandard, einen `TCP-` oder `BluetoothServerConnectionHandler` der den Server-Thread implementiert. Der Server-Thread wartet auf eingehende Verbindungen von Clients und erzeugt für jede Verbindung ein `Connection`-Objekt. Jede `Connection` startet wiederum zwei Threads. Eine Thread-Loop wartet im blockierenden `read()`-Aufruf eingehende Daten, während der andere Thread das Senden der in der Message-Queue eingereihten Nachrichten koordiniert und dadurch mehrfaches gleichzeitiges Senden auf die Verbindung verhindert. Die folgende Abbildung 4 zeigt den vereinfachten Ablauf von Start bis Ende einer Prozess-Laufzeit.

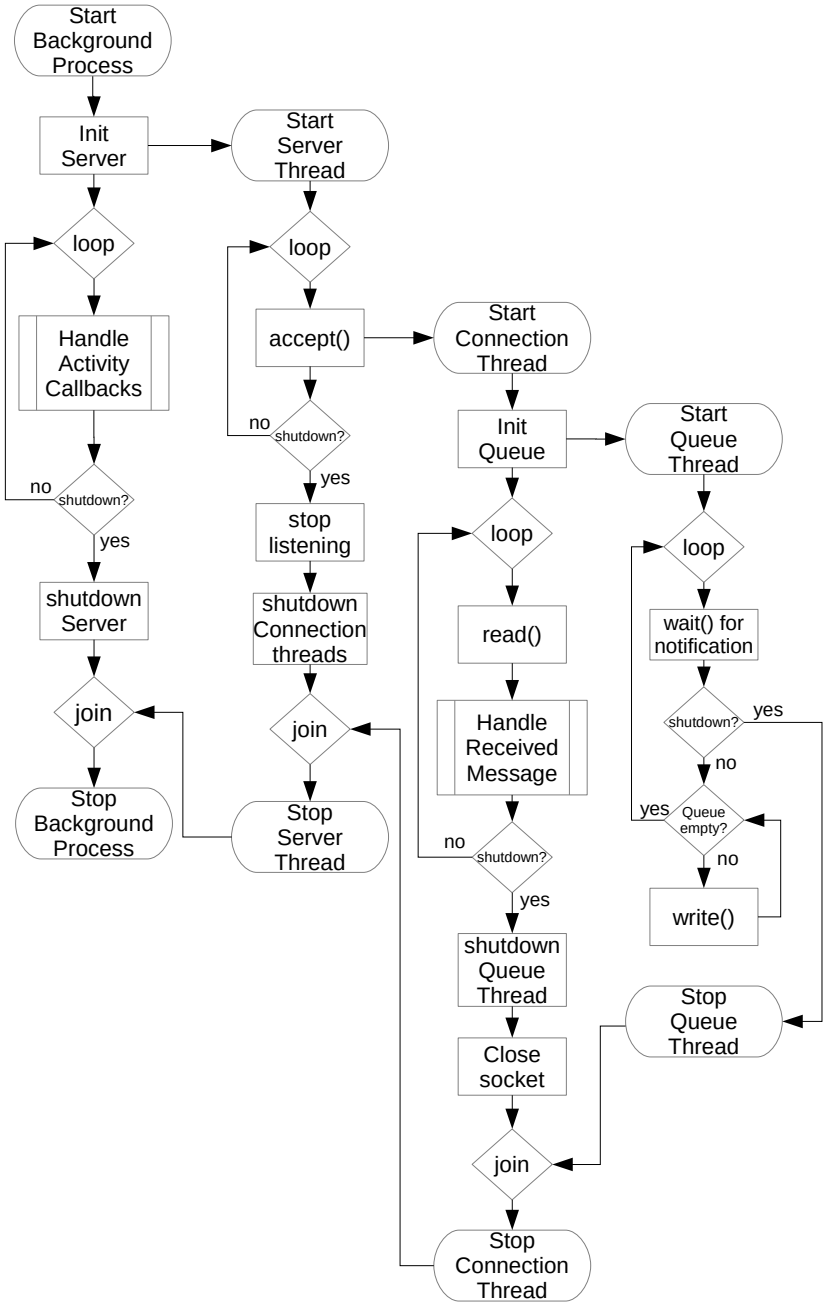

Abbildung 4: Flussdiagramm des Server-Services

4.2 Client-Service

Der Hintergrund-Prozess der Client-App erzeugt zunächst einen Client-Handler und wählt hierfür, je nach genutztem Übertragungsstandard, den `TCP-` oder `BluetoothClientConnection-Handler`. Anschließend wird von der Activity der Befehl zum Server-Discovery erwartet, den der `ClientConnectionHandler` durchführt und die gefundenen Server der Activity übermittelt. Der Anwender kann nun in einer Liste den gewünschten Server auswählen. Der `ClientConnection-Handler` versucht nun, sich zu diesem Server zu verbinden und erzeugt dafür wie die Server-App ein entsprechendes `Connection`-Objekt. Da die `Connection`-Klasse bei Server und Client verwendet wird, verhält sich das Objekt wie bereits beim Server beschrieben.

Die folgende Abbildung 5 veranschaulicht den Ablauf des Client-Prozesses während seiner Laufzeit.

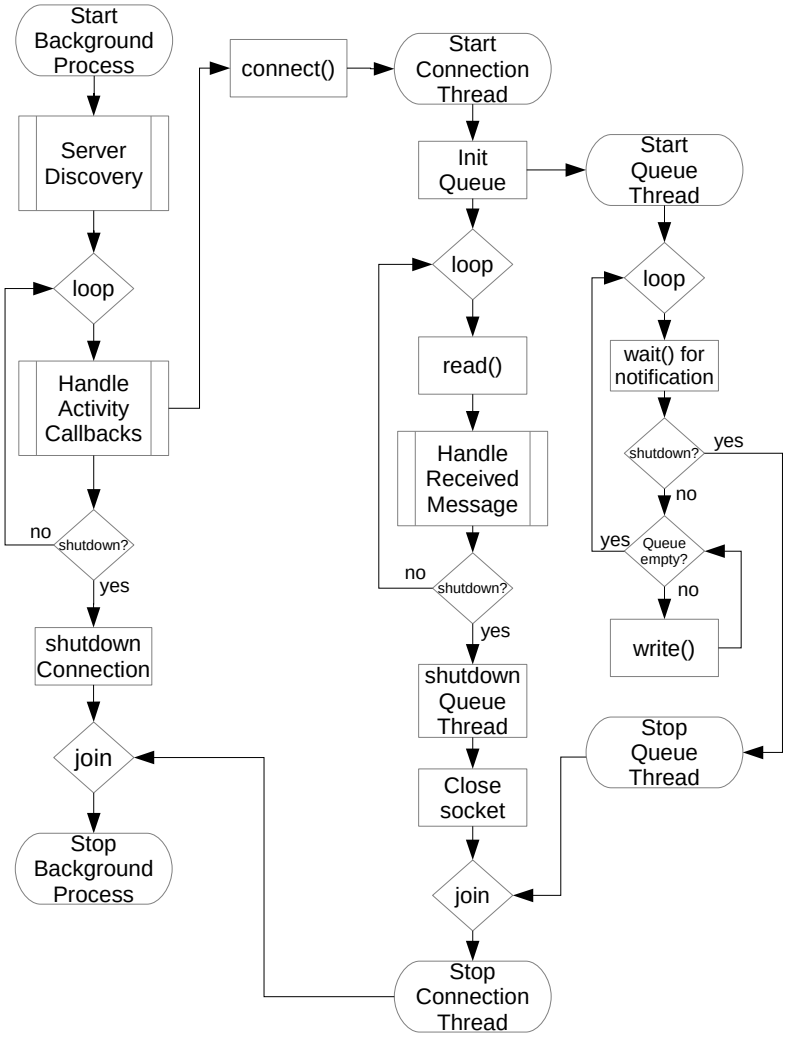

Abbildung 5: Flussdiagramm des Client-Services

5 Anleitung

5.1 Server-App

Der Server-Dienst zum Annehmen der Client-Verbindungen ist direkt mit dem Start der Server-App verfügbar und muss nicht separat gestartet werden. Der Frontend-Teil der App beinhaltet die beiden Bildschirmseiten „Playlist“ und „Clients“.

Auf der erstgenannten Seite kann die globale Playlist eingesehen werden und die Steuerung der Wiedergabe erfolgen (Abbildung 6). „Start“, „Pause“, „Stop“, „Next“ sind die verfügbaren Steuerungsoptionen. Außerdem wird während der Wiedergabe eine Seekbar angezeigt, die den Fortschritt der Wiedergabe grafisch darstellt, aber auch zum Sprung an eine bestimmte Position dienen kann. Die Playlist wird automatisch aktualisiert, sobald eine Änderung der jeweiligen Client-Playlist erfolgt.

Die Bildschirmseite „Clients“ zeigt Alias-Name und die IP-Adresse bei TCP- bzw. die Mac-Adresse bei Bluetooth-Übertrang der verbundenen Client-Geräte an (Abbildung 7). Über das Optionen-Menü lässt sich die Einstellungsseite aufrufen oder die App beenden. Die verfügbaren Einstellungen sind in Abbildung 8 ersichtlich.

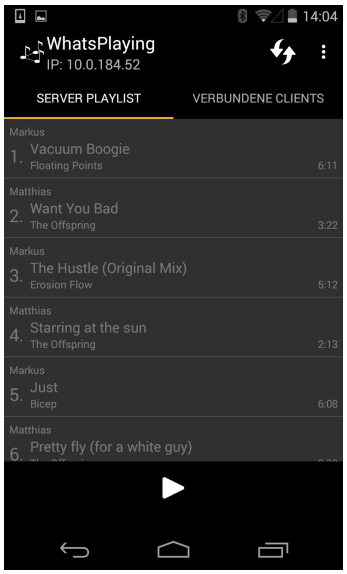

Abbildung 6: Server: Globale Playliste

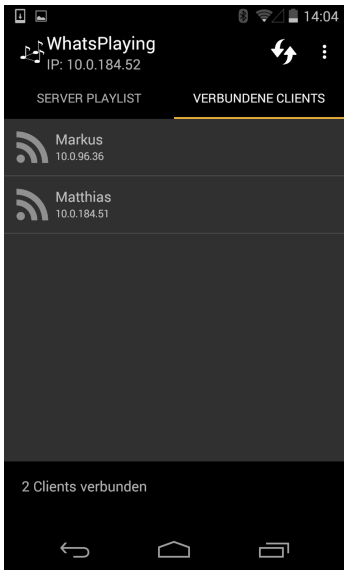

Abbildung 7: Server: Clientliste

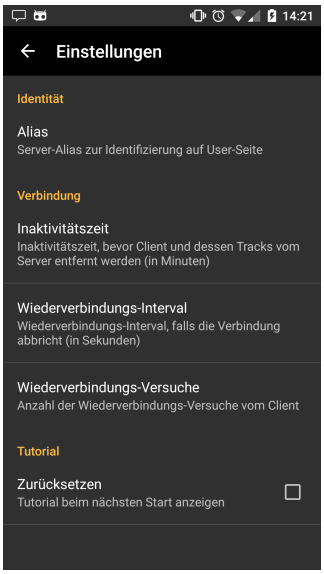

Abbildung 8: Server: Einstellungen

5.2 Client-App

Auf Clientseite bietet die App dem Benutzer die vier Bildschirmseiten „Aktive Server“, „Musik-Bibliothek“, „Lokale Playlist“ und „Globale Playlist“. Es werden je nach Zustand nicht alle Seiten zeitgleich angezeigt. Die Serverliste ist nur verfügbar, wenn keine Verbindung aktiv ist, während die globale Playlist umgekehrt nur bei aktiver Verbindung dargestellt wird.

Auf der ersten Seite (Abbildung 10) werden alle verfügbaren Server dargestellt. Der Server sendet ein Magic-Packet per UDP-Broadcast oder stellt einen Bluetooth-Dienst bereit und wird so von den Clients selbstständig erkannt und in der Serverliste angezeigt. Alternativ ist über das Optionen-Menü eine manuelle Angabe der IP-Adresse möglich. Auf Bluetooth-Seite müssen die Geräte jedoch zunächst gepaart werden.

Wie schon die Server-App verfügt die Client-App ebenfalls über eine Einstellungsseite mit grundlegenden Konfigurationsmöglichkeiten (Abbildung 11).

Abbildung 9: Client: ShowCaseView

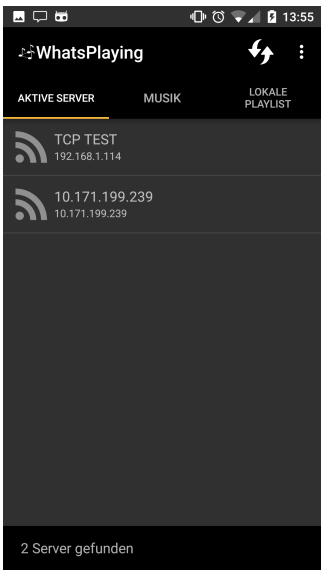

Abbildung 10: Client: Serverliste

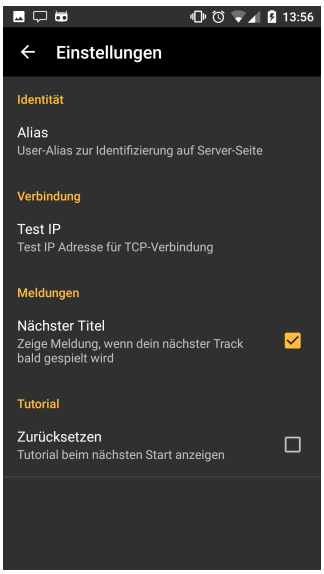

Abbildung 11: Client: Einstellungen

Auf der Seite „Musik-Bibliothek“ (Abbildung 12) kann die lokale Musik-Sammlung durchsucht werden. Es ist sowohl möglich, die Liste mit Hilfe des Suchfelds am unteren Seitenrand zu durchsuchen (Interpret, Titel, Dateiname), als auch die Liste nach Interpret, Titel oder Dauer zu sortieren. Um gewünschte Musikstücke in die lokale Playlist zu übernehmen, hakt man einfach die jeweilige Checkbox an. Auch eine Abwahl ist möglich.

Die Seite „Lokale Playlist“ (Abbildung 13) beinhaltet eine Liste der ausgewählten Musikstücke und eine einfache Playlistverwaltung am unteren Seitenrand, um diese Speichern, Laden und löschen zu können. Auch hier können Musikstücke über das Mülleimer-Symbol aus der aktuellen Zusammenstellung entfernt werden.

Eine Kopie der serverseitigen, globalen Playlist ist auf der Seite „Globale Playlist“ (Abbildung 14) einsehbar. Dabei werden auch Status-bezogene Angaben dargestellt, wie z.B. ob der Track gerade übertragen, abgespielt oder für die Wiedergabe gesperrt ist.

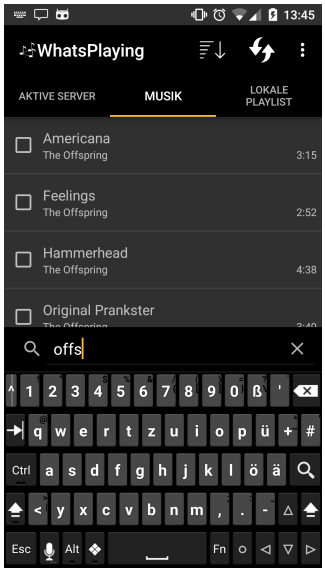

Abbildung 12: Client: Musik-Bibliothek

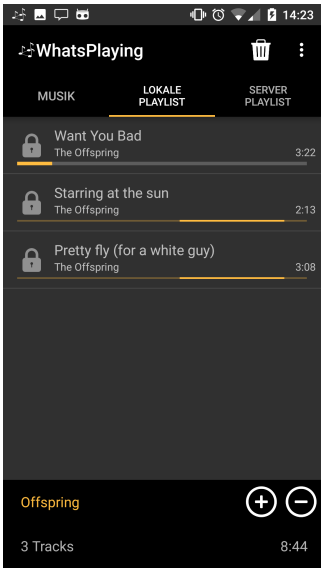

Abbildung 13: Client: Lokale Playlist

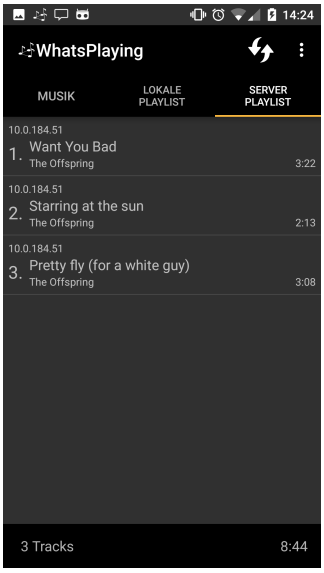

Abbildung 14: Client: Globale Playlist

6 Probleme

Bei der Entwicklung der App kam es an einigen Stellen zu mehr oder weniger komplexen Problemen, die im Folgenden genauer erläutert werden.

6.1 Bluetooth

Aufgrund mehrerer Probleme bei der Verwendung von Bluetooth und der geringen Übertragungsrate wurde im späteren Verlauf des Entwicklungsprozesses auf die Kommunikation mittels TCP (sprich WLAN) gewechselt. Die Kernprobleme beim Einsatz von Bluetooth werden im Folgenden kurz erläutert.

6.1.1 Energiesparmodus

Bei Verwendung des *Logical Link Control and Adaptation Layer Protocols* (kurz L2CAP, Standard für Android *Bluetooth Sockets*) zur Übertragung der Musikdateien traten beim Empfänger reproduzierbare Fehlverhalten auf, welche ebenfalls beim Sender zu einer vollständigen Blockade des Programmflusses führten.

Beim Empfang größerer Dateien, bei denen die Übertragung einige Sekunden Zeit in Anspruch nimmt, blockiert nach einer Weile der Empfang (das Auslesen des Eingangspuffers), weshalb der Sender aufgrund der Flusskontrolle nicht mehr weiter senden kann und ebenfalls blockiert.

Nach längerer Fehlersuche, konnte die Ursache auf den Energiesparmodus der Bluetooth-Controller eingegrenzt werden. Wenn über den Controller einige Sekunden keine Daten gesendet werden, schaltet Android oder der Treiber den Controller in den Energiesparmodus, auch wenn gerade Daten empfangen werden. Durch Abschalten des physical layers läuft dann der Eingangspuffer leer, wodurch Sender und Empfänger in ihren jeweiligen `write()` und `read()` Aufrufen blockieren.

Ein Workaround, bei dem alle 5 Sekunden ein Keep-Alive-Byte zum Sender übertragen wird, löst dieses Problem. Durch das Senden wird der Idle-Timer des Controller stets zurückgesetzt und bleibt deshalb aktiv.

6.1.2 Byte Order

Mit der Android-Version 6.0.1 werden die UUIDs der selbst definierten Services in umgekehrten Byte-Order übermittelt. Ob es sich hierbei um einen Bug von Android handelt ist nicht klar.

6.1.3 Pairing

Bevor ein neuer Bluetooth-Teilnehmer die Service UUIDs eines anderen Bluetooth-Teilnehmers über das *Service Discovery Protocol* (kurz SDP) ermitteln kann, ist es notwendig, dass sich beide Geräte gekoppelt haben. In der Praxis würde dies jedoch sicherlich zu Akzeptanz-Problemen führen, da das zentrale Smartphone für den Kopplungsvorgang mit jedem neu hinzukommenden Teilnehmer bedient werden muss.

6.2 TCP

Ein Abbruch der Verbindung, beispielsweise durch plötzlichen Ausfall des WLAN-Access-Points, kann bei Verwendung von TCP nicht sofort vom Server erkannt werden. Ein ordentlicher Verbindungsabbau per FIN-Paket wird nicht eingeleitet. Stattdessen bemerkt der Server dies erst, wenn entweder der Versuch auf den Socket zu schreiben fehlschlägt, oder ein vom Betriebssystem initiierte Keep-Alive-Nachricht mit sehr hohem Timeout (in der Regel über 60 Minuten) nicht beantwortet wird. Das frühzeitige Erkennen eines Verbindungsabbruchs ist jedoch für das

Entfernen der Titel des betroffenen Clients aus der globalen Playlist wünschenswert.

Zur Lösung des Problems muss ein Mechanismus implementiert werden, der in bestimmten kurzen Zyklen eigene Keep-Alive-Nachrichten an die Clients sendet.

6.3 Algorithmus zur Playlist-Verwaltung

Die höchste Komplexität steckt im Algorithmus zur Verwaltung der globalen Playlist. Der Algorithmus muss aus allen lokalen Playlisten der verbundenen Clients stets eine konsistente globale Playlist erzeugen und diese mit den Clients synchronisieren. Dabei müssen mögliche Modifikationen der lokalen Playlisten, wie das Hinzufügen und Entfernen von Musikstücken, ständig mit der globalen Playlist synchronisiert werden.

Der Algorithmus muss ebenfalls die für das Buffering vorgemerkten Dateien bei den entsprechenden Clients für Playlist-Modifikationen sperren.

7 Ausblick

Bedingt durch die begrenzte zur Verfügung stehenden Arbeitszeit, konnten nicht alle Funktionen in der gewünschten Tiefe implementiert werden.

Da zur Zeit der Java-Serialisierungs-Algorithmus zur Serialisierung von Nachrichten-Objekten genutzt wird, wäre die Entwicklung eines eigenen Übertragungsprotokolls mit einem standardisierten Serialisierungs-Algorithmus wünschenswert. Nur so könnten Server- und Client-Anwendungen in anderen Programmiersprachen für andere Betriebssysteme (wie iOS oder Windows) zuverlässig realisierbar gemacht werden.

Für die Android Implementierung wäre ebenfalls eine WLAN Hot-Spot Funktion denkbar, die es ermöglicht einen Server ohne zusätzliche WLAN Hardware zu starten. Für diese Funktion wäre jedoch das Unterbinden des Routings in das mobile Datennetz eine für die Praxis notwendige Bedingung, da sonst alle per WLAN verbundenen Geräte das Datenvolumen des mobilen Datentarifs des Servers verwenden könnten.

Ein Bewertungssystem, bei dem jeder Client eine Stimme zu jedem der in der globalen Playlist verfügbaren Musik-Titeln abgeben kann, wäre ebenfalls denkbar. So könnten Titel, die einen beim Server konfigurierbaren Schwellwert an negativen Stimmen erreicht haben aus der Liste entfernt werden, oder Titel mit einer gewissen Menge an positiven Stimmen vorzeitig oder häufiger abgespielt werden.