# A Robot Economy for Music Without Any Intermediaries

Tim Wissel

TU Delft

# A Robot Economy for Music Without Any Intermediaries

by

Tim Wissel

to obtain the degree of Master of Science
at the Delft University of Technology,
to be defended publicly on XXX at XXX.

Student number: 4460251
Project duration: March 6, 2020 – XXX
Thesis committee: Dr. ir. J. Pouwelse, TU Delft, supervisor

An electronic version of this thesis is available at `http://repository.tudelft.nl/`.

TU Delft

# Preface

Preface…

*Tim Wissel*
*Delft, March 2020*

# Contents


1 Introduction 1

2 Problem description 3
- 2.0.1 Platformization and gatekeeping. . . . 3
- 2.0.2 Intermediaries take a large share. . . . 3
- 2.0.3 Monopsony power . . . . 4
- 2.0.4 Recommendations and curatorial power. . . . 4

3 State of the Art 5
- 3.1 Decentralized audio streaming services. . . . 5
- 3.2 TrustChain-Superapp . . . . 5
- 3.3 Tribler and bandwidth tokens. . . . 5
- 3.4 Decentralized content delivery networks . . . . 5
- 3.5 Incentives for file spreading. . . . 7

4 Design 9
- 4.1 Phone-to-phone connectivity. . . . 9
- 4.2 Release model. . . . 10
- 4.3 Identity and authenticity . . . . 10
- 4.4 Distributed storage . . . . 10
- 4.5 Distributed downloading and uploading . . . . 11
- 4.6 Distributed search algorithm . . . . 11
- 4.7 Direct payments without intermediaries . . . . 11
  - 4.7.1 Wallet . . . . 11
  - 4.7.2 Artist Income Division Algorithm . . . . 11

5 Implementation 13
- 5.1 Features overview. . . . 13
- 5.2 Playlist overview interface . . . . 14
- 5.3 Playlist fragment interface . . . . 14
- 5.4 Wallet interface . . . . 14
- 5.5 Networking . . . . 15
  - 5.5.1 Release creation and sharing. . . . 15
  - 5.5.2 Content seeding . . . . 15
- 5.6 Caching. . . . 15
- 5.7 Music Player and Streaming. . . . 17
  - 5.7.1 Priority handling. . . . 17
- 5.8 Identity and authenticity . . . . 17
  - 5.8.1 Keyword search . . . . 17
- 5.9 Donations and payments . . . . 17
- 5.10 Quality assurance. . . . 17

6 Conclusion 19

Bibliography 21

# 1

# Introduction

Most audio streaming services are run by companies, incentivized to make money. They take large cuts of the subscription money from its users. As a result, the artists receive a low compensation. The distributors Spotify, iTunes and Google Play take on average a 25% cut for signed records and 40% cut for unsigned records. This thesis investigates the feasibility and usability of an audio streaming service without a central distributor.

This thesis proposes a solution in the form of a decentralized system which uses a decentralized autonomous organization (DAO) to operate. Listeners, artists and robots form this DAO. The DAO has a shared responsibility for distributing content. In this system, its users (artists and listeners) share audio files and metadata without any middleman. Additionally, users can give donations to artists while the system does not take a cut of these donations. The user can use this system to discover, search and play audio files, targeted at music and podcasts.

Section X describes the design of the system, section Y its implementation and Z its testing results.

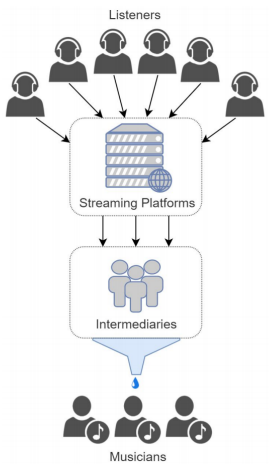


Figure 1.1: Artist compensation inconsistency

# 2

# Problem description

The most widely used music streaming services, with the largest music catalogs, run centralized, proprietary and closed-source software. The companies owning these services have a large amount of power in the music streaming industry because of their scale. The top 5 streaming services have a combined market share of 81%, so this can be regarded as an oligarchy. Because of their power, they can ask high commission fees or lock artists to one platform. As a result, artists receive low compensation. Furthermore, the recommendation and playlist generation algorithms are also a black box for the user. This gives streaming companies curatorial power. At the time of writing, there exists no alternative decentralized and transparent music streaming system with peer-to-peer payments directly to artists. *How can we design and implement a music streaming service that distributes the power from one authority to its users?*

### 2.0.1. Platformization and gatekeeping

The infrastructure of current internet applications are increasingly moving towards 'platformization'. In essence, platforms are taking control of "the surface on which the market exchange take place" (Andersson Schwarz, 2016) with digital distribution and network effects enabling an increasing centralization of power. This phenomenon is related to IT gatekeeping: tying access of content to a specific internet service. An example of this is the release of the album *The Life of Pablo* in 2016, which was contracted to only be played on one platform, Tidal. In addition, the big music platforms are able to create a digital enclosure of content. This facilitates rent-based monetization of user data, described by Harvey (2002) as 'monopoly rent'. Namely, advertisers can lease a virtual space, to match their advertisements to user profiles. The latest movement in platform accumulation is the monopolization of data. Large scale of data about user interactions with the platform forms a 'monopoly of knowledge' (Innis, 2007). The power of platform companies are raising because platforms, in general, tend towards monopoly (Srnicek, 2017).

In relation to gatekeeping, platforms are now given the task to perform moral judgements on content, for example whether to censor a certain artist. This is controversial as these judgements are no longer in the hands of users but rather in the hands of companies. Recent examples exist such as the disappearance of Li Zhi[1], who published songs about democracy and social issues in China. All of China's main streaming sites removed his songs. In 2019, Apple Music also removed content from their platform by singer Jacky Cheung, who referenced the tragedies of Tiananmen Square in his songs[2].

### 2.0.2. Intermediaries take a large share

Artists publishing their content on a music streaming service such as Google Music, Spotify and Apple Music receive low compensation, because the intermediaries take a large share. According to Midia Research, the top 5 streaming services control 81% market share (Mulligan, 2020), which can be regarded as an oligarchy. Together these streaming services have the power to ask high subscription fees. After multiple consecutive years of growth, 2019 became the first year in which digital streaming is the single biggest source of revenue for the music industry globally (IFPI, 2020). The global music market also gained a 33.5% growth in paid

[1] https://www.independent.co.uk/news/world/asia/tiananmen-square-china-li-zhi-singer-disappears-anniversary-protests-a8940641.html

[2] https://hongkongfp.com/2019/04/09/apple-music-china-removes-jacky-cheung-song-reference-tiananmen-massacre/

streaming subscribers during this year (IFPI, 2020). This trend is shifting power from decentralized music stores to centralized streaming services.

At the same time, streaming services take a large cut in revenue, and artists are having a harder time making money from music. According to investigations by aCooke (2018) and ReCode (2015), the revenue cut of Apple Music and Spotify is between 25% and 30%. An additional problem is opacity: streaming deals on these platforms remain behind closed doors. For these reasons, massive audiences are needed to generate sustaining profits. An investigation by Bloomberg[3] shows that a whopping 152,094 Spotify subscriber streams generate $100 on average for artists. Consequently, only 0.733% of all acts generate enough revenue for an artist to make a living (Ingham, 2018). The International Federation of the Phonographic Industry states that as of 2018 there exists a "value gap" in digital music streaming, meaning a “mismatch between the value that some digital platforms [...] extract from music and the revenue returned to the music community–those who are creating and investing in music” (IFPI, 2018).

### 2.0.3. Monopsony power

Monopsony power means that a dominant buyer has the power to push prices down with suppliers. In the context of music, this means that artists have little choice over which platform to publish their music on, because of the dominance of one platform. A few major players in the music industry together form an oligopolistic market. Monopsony power in this area can lead to squeezing the producer side. An example of monopsony power is an event that happened in 2014, between Amazon and Hachette. Amazon, having a large market share on e-books, used its commercial muscle to demand a larger cut of the price of Hachette books it sells. This included for all Hachette books “preventing customers from being able to pre-order titles, reducing the discounts it offered on books and delaying shipment” (Ellis-Petersen, 2014).

Along the same lines, the music streaming oligarchs can use their commercial muscle to demand low pays to artists. Spotify founder and CEO Daniel Ek declared to its investors that the increase in interactions with its in-house curated playlists “puts Spotify in control of the demand curve” [4].

### 2.0.4. Recommendations and curatorial power

The Big Tech music companies recommend content that best fits their business model, which may be contrary to what is most useful to their customer. The companies can promote or dis-promote content by their choosing. This shows “curatorial power”: the ability to advance own interests by organizing and prioritizing content (Prey, 2020).

Musicians and record labels are increasingly more dependent on landing on Spotify-curated playlists. For example, a study done by the European Commission shows that, for a track to land on the Spotify-curated playlist “Today’s Top Hits”, it will see an increase of $163,000 in revenue (Aguiar & Waldfogel, 2018). 99 of the top 100 playlists are curated by the streaming company. So its recommendation algorithms and playlist curation systems are highly influential.

As the top music streaming services run closed-source software, the inner workings of the recommendation algorithms are opaque to the user. These algorithms, fed by user interaction data, are in some extend also a black box to the company, as they are built using machine learning technologies. However, as noted by (Gillespie, 2014), the impression that algorithms are objective is a “carefully crafted fiction”. Namely, they are altered based on company strategies. Companies are not obliged to explain their algorithms. In the context of recommendations, this leads to a “threat of invisibility": the problem of content regularly disappearing (Bucher, 2018), a phenomenon which is out of the hands of the artist, because of an asymmetry in knowledge over the algorithm workings. Frustrating for artists and labels is also the opaqueness of getting playlisted: it is unclear why “[...] a particular track was placed, or replaced, on a playlist” (Prey, 2020).

[3] https://www.bloomberg.com/opinion/articles/2017-09-25/the-music-business-is-more-unfair-than-ever

[4] https://investors.spotify.com/financials/press-release-details/2019/Spotify-Technology-SA-Announces-Financial-Results-for-Fourth-Quarter-2018/default.aspx

# 3

# State of the Art

## 3.1. Decentralized audio streaming services

Multiple decentralized audio streaming services exist. Examples are Audius[1], Resonate**?** and eMusic[2]. All of these systems have in common that they save metadata and identifiers of audio files on a blockchain, and save the audio files in an off-chain database. All these off-chain databases are structured like IPFS[3] with a company-run centralized interface between the user interface and the database. For a system to be fully decentralized, this layer should be removed. These solutions are closed source. Moreover, they use their own cryptocurrency to pay their artists which is an unstable income.

## 3.2. TrustChain-Superapp

The TrustChain superapp (Skala, 2020) is a framework for implementing mobile Android decentralized applications. It allows for storing append-only immutable data on TrustChain (Otte, de Vos, & Pouwelse, 2020) and spreading this data in a phone-to-phone serverless network. It follows the concept of super apps (Huang, 2019), meaning that it contains many mini-apps which use the same networking interface. The Superapp is an Android app as seen in fig. 3.1. Its mini-apps implement decentralized democratic voting and has bitcoin payment integration, among other features.

## 3.3. Tribler and bandwidth tokens

Tribler is a peer-to-peer system to share, download and stream multimedia. It has implementations for desktop environments and an Android prototype. It makes use of BitTorrent for file transfer and adds anonymization techniques on top of it. In addition, it makes use of its bandwidth tokens: an incentive system to increase cooperation between users, in order to achieve high availability of downloads. In essence, it subtracts tokens for downloading content from peers and rewards tokens for helping peers. An overview of the Tribler desktop interface can be seen in fig. 3.2.

## 3.4. Decentralized content delivery networks

Decentralized content delivery networks are being investigated by multiple systems such as VideoCoin[4] and DCDN[5]. Most of these start-ups use blockchain technology and their own-released cryptocurrency as a token to pay nodes that serve the content. This means that the incentive for running a node depends on the value of those cryptocurrencies, so this is an unstable situation for workers.

A fully decentralized audio streaming service requires sharing and streaming audio files over a network of nodes in which any participant can start and run a node. An example of such network is BitTorrent. The challenge with BitTorrent acting as a streaming service is that the requirement from the user perspective is to

[1] `https://audius.co`
[2] `https://eMusic.com`
[3] `https://ipfs.io/`
[4] `www.videocoin.io`
[5] `https://www.dcdn.com/`

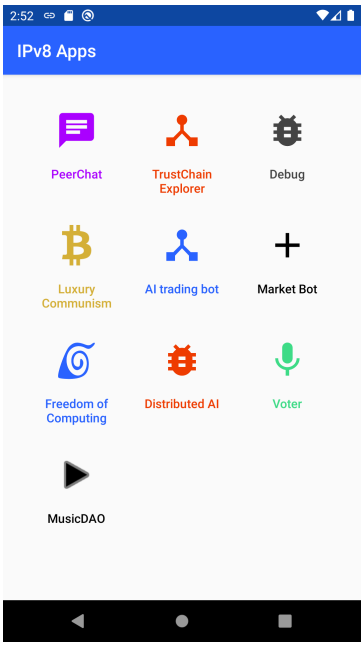


Figure 3.1: Trustchain-Superapp overview

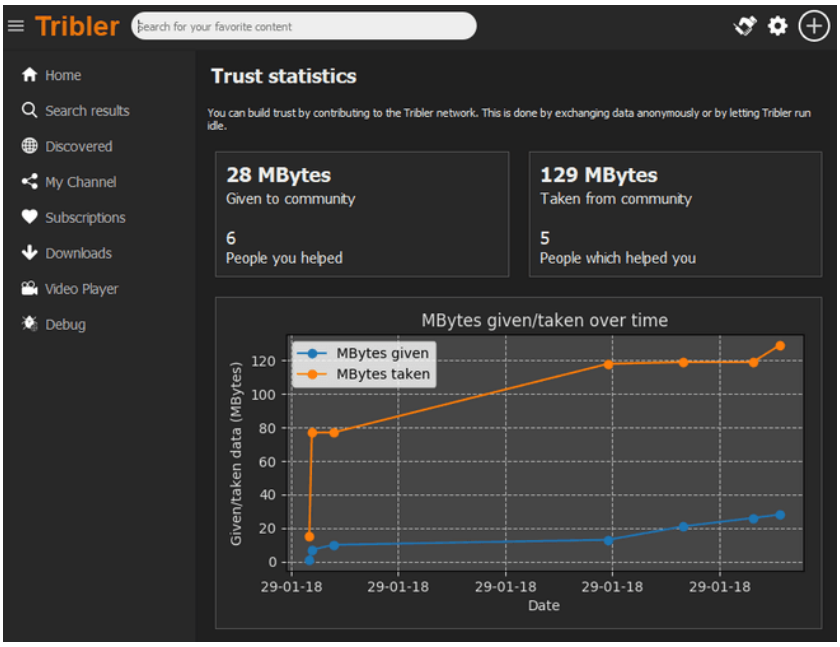


Figure 3.2: Tribler desktop interface, showing the bandwidth incentive system overview

have low latency for streaming and buffering media files. For each file, the peer discovery algorithm is run, which is a slow-start algorithm. It also relies on having enough seeders per file available.

Torrent files contain a list of chunks, which represent the different parts of the related file. These chunks are called torrent pieces. Flawless streaming of media files over BitTorrent requires a smart algorithm to predict what file is requested next, and what torrent pieces should be loaded. BitTorrent relies on trackers to perform peer discovery. However, trackers are a central point of failure. To make the system more decentralized, a solution using independent trackers and a gossip protocol**?** can be used.

## 3.5. Incentives for file spreading

In a DAO, the party responsible for hosting and spreading of files is not well-defined. To tackle the tragedy of the commons, entities should be incentivised just enough for the system to be sustainable and usable, but no more. An example incentive system is bandwidth tokens (de Vos & Pouwelse, 2018).

# 4

# Design

In this section we present the design of our software application MusicDAO. MusicDAO is a mobile music streaming and discovery app, with peer-to-peer payment to artists in the form of donations and subscriptions. The MusicDAO is fully decentralized by design. This means there are no intermediaries, third parties or proprietary servers needed. All users of the app form a community to share audio tracks and transfer money. Any user can join this community, publish their musical works and receive money from its listeners. With the goal of distributing power in the music industry, every peer in the network will have the same rights and access to the same functions.

The overall design of the system can be seen in fig. 4.1. This describes the interaction between the different components, libraries and frameworks. The following sections explain the designed features, components and design choices.

## 4.1. Phone-to-phone connectivity

Mobile phones have the largest share of any type of device using music streaming services. If the software that we design and implement runs well on a network consisting of only mobile devices, we can conclude that it will run well on better hardware (such as PCs) as well. For these reasons, we design our system to work on mobile phones only.

We choose to use the Android framework for decentralized apps as proposed by Skala (2020). This gives us a toolbox for building phone-to-phone serverless apps and make use of its distributed ledger technology to build a public database. It also gives us a public-key infrastructure to identify and authenticate artists.

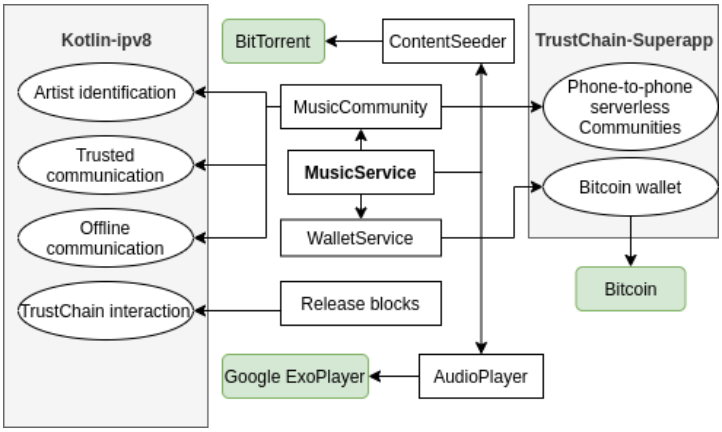


Figure 4.1: Architecture overview, with in green the external libraries

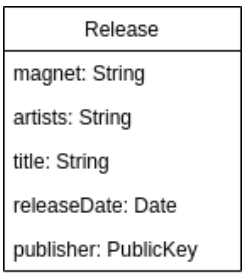


Figure 4.2: Release blocks structure as seen on TrustChain

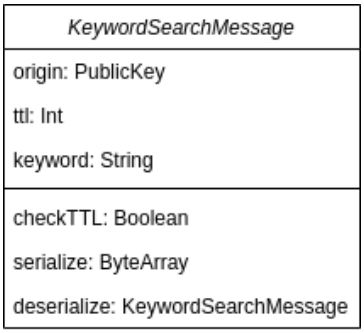


Figure 4.3: KeywordSearchMessage object sent over IPv8 in MusicCommunity

## 4.2. Release model

Central to our system will be the storage of metadata and audio files for playlists. We design an abstract model for the structure of this metadata, so that the artist is free in the way to release music content. The artist may publish tracks as part of a single, an EP, an album or any other structured list of tracks.

A Release is an object describing a list of tracks that are published by a clearly identifiable artist or group of artists. It is modeled as shown in fig. 4.2. Release objects are shared between peers in the network. By discovering many of those objects, a user can see and browse through them to select a track to play. A Release object merely contains metadata of the tracks. We design the network to have a separate channel for downloading the track files. This is to enable fast discovery and searching of Releases, as Release objects have a small byte size.

## 4.3. Identity and authenticity

As we design a system that is fully decentralized, we cannot use a central database to record user identities. Therefore every user should generate a unique identity to be used in the network, and must be able to give proof of this identity. We use a public key infrastructure (PKI) which achieve these goals. Every user stores their private and public key on their device, and only share their public key. The keypair has a mathematical property that allows verification of messages that are signed with a private key. By comparing the public key of a peer with their signed message, anyone can verify the authenticity of the message.

In the context of MusicDAO we use this PKI to proof ownership of Release objects. All Release objects are signed using the owner's private key and the signature is added to the object. Any user receiving this object can verify its authenticity.

We choose to use the public key infrastructure as implemented in the TrustChain-Superapp (Skala, 2020). This gives us a unique identity per Android phone, by abstracting the network identities such as (changing) IP addresses and Bluetooth addresses.

## 4.4. Distributed storage

Central to our system is sharing downloading and storage of audio files and Release objects (see 4.2). To design a system which has no middlemen or regulators for publishing Releases, and has no central control, a distributed storage system is required. This storage system should have the following properties: immutability (data cannot be tampered with), resiliency (data should be available as long as users want it) and rigorous duplication (all objects should live on multiple devices so they do not disappear when requested). Distributed ledger technology (DLT) allows for these properties, so we design our system with a DLT as a major component.

One implementation of this technology is TrustChain (Otte et al., 2020) which allows for recording transactions between peers in a linearly scaling public ledger. Every peer has its own immutable and public blockchain which shows its history of transactions with others. This way we can establish trust in a certain party. In our context, we can use this mechanism to estimate trust in artists by inspecting their history of uploads. We choose TrustChain because of its scalability and trust mechanism, and because it has a native implementation for Android, as described in 3.2. In addition, it allows for offline communication, so users can download and explore new content using Bluetooth or LAN.

## 4.5. Distributed downloading and uploading

To be able to have low latency for discovering and playing music tracks, while using no central nor high-throughput servers, the network demands participants to upload content continuously. We design the app to, by default, use the network capabilities of the mobile device to upload content as much as possible. This is constrained by networking hardware, data subscription plans and other software running on the phone.

## 4.6. Distributed search algorithm

For searching content we use a simple distributed algorithm called XYZ (cite). Pseudocode of this algorithm is shown in fig. X. It asks peers around for content tagged with some keyword. When a peer finds a match on their local database, it sends this Release object to the original asking peer. Otherwise it forwards the query to their neighbours, after reducing the time-to-live property by 1. The messages stop being forwarded once their time-to-live property hits below 1. The structure of search messages are shown in fig. 4.3.

## 4.7. Direct payments without intermediaries

As we are designing a system with no intermediaries, it should be possible to give money directly to artists. Cryptocurrency allows for peer-to-peer payments which achieve this goal, so we use this in the MusicDAO. Cryptocurrency payments will be used for two different functionalities: a user can send a donation to an artist, or a user can pay artists using a monthly subscription system. This subscription system pays artists that the user listened to, using the Artist Income Division Algorithm (see 4.7.2).

We choose to use Bitcoin as a cryptocurrency as it has shown to be a secure and popular payment system, and it does not rely on any third parties to run. It also allows for making a experimentation environment without any high-throughput external servers.

### 4.7.1. Wallet

Cryptocurrency implementations allow for private/public key-pairs which can be interpreted as a kind of wallet; the funds can only be unlocked by a holder of the private key. In the case of MusicDAO we design the app to include a wallet for every user. To receive money, every artist should share their public key to all of their listeners. To achieve this, the public key of their cryptocurrency wallet is included as a property of the Release objects (see 4.2). As there are no institutions or banks involved in storing money, users will be required to keep their private key safe.

### 4.7.2. Artist Income Division Algorithm

To provide a stable income for artists, in the form of reoccurring payments, we design the Artist Income Division Algorithm. This algorithm calculates how periodic subscription money is split into payments to artists. The user can enable a periodic payment. This money is then divided over the artists the user listens to, in proportion to the amount of interaction with each artist. Interaction can be measured in e.g. time listened, plays or feedback in the form of likes. The details of this division is explained in the implementation section of AIDA (see X).

This sec may be removed

# 5

# Implementation

The application is implemented as a 'mini-app' of the *TrustChain Superapp* (Skala, 2020). This follows the concept of super apps (Huang, 2019). The app is published on the Android Play Store[1] and runs on Android 5.1 and above. Its code is publicly available[2]. As programming language Kotlin is selected, as it is the preferred language for Android development (Haase, 2019). Moreover the underlying technology stack is also written in Kotlin, so this allows for neat integration. This section describes the implementation choices, usage of external libraries and presents the user interface of MusicDAO. The main interfaces of the app can be seen below (figs. 5.1, 5.2 and 5.6). An overview of the structure of the code can be seen in the package diagram in fig. 5.10.

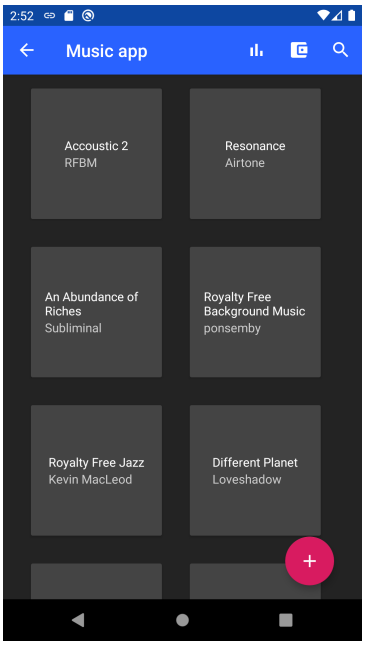


Figure 5.1: The playlist overview screen, which is the entrance screen

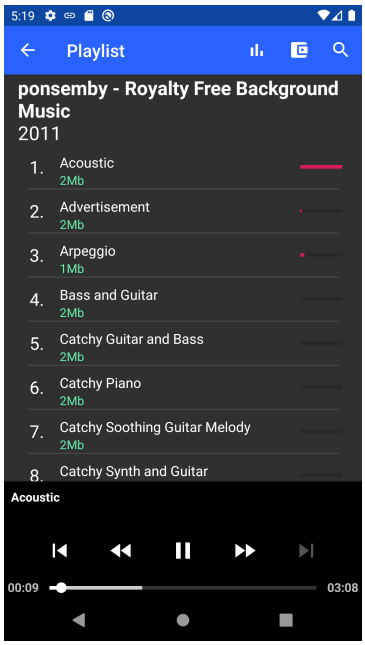


Figure 5.2: Playlist fragment, showing all tracks of one Release

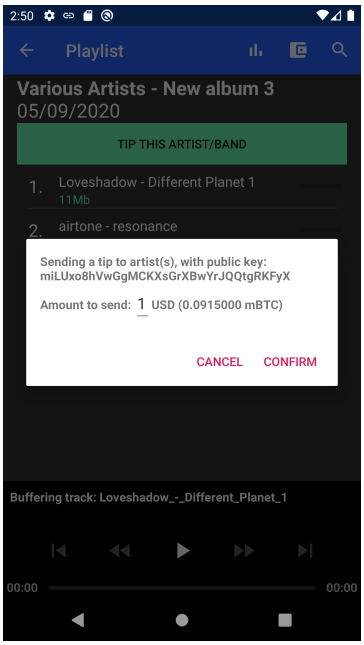


Figure 5.3: Sending a tip to an artist or band

## 5.1. Features overview

We implemented a peer-to-peer system to share, discover and listen to music, and to give payments to artists. The system requires no servers, middlemen or third parties to run. Our Android app contains the following features:

[1] `https://play.google.com/store/apps/details?id=nl.tudelft.trustchain`
[2] `https://github.com/Tim-W/trustchain-superapp`

| Name | Version | Usage |
|---|---|---|
| JLibtorrent | 1.2.10 | Peer-to-peer file distribution |
| TorrentStream-Android | 2.7.0 | Video streaming library |
| ExoPlayer | 2.10.5 | Android multimedia player |
| BitcoinJ | 0.15.7 | Java bitcoin interface |
| XChange | 5.0.1 | Crypto/USD price conversion |

Table 5.1: Notable libraries used

- Defining album/single/EP releases using metadata, and sharing those with peers; Sharing of your audio tracks with peers; Immutable storage of music metadata and cryptographic identification of artists.
- Streaming music over BitTorrent; Prioritization algorithm to minimize streaming latency; Caching audio files and metadata.
- Browsing playlists; Local and remote keyword search for music.
- Mobile bitcoin wallet implementation; Peer-to-peer donations to artists using Bitcoin.

The following planned features were not implemented:

- Artist Income Division Algorithm.
- Content sorting algorithm based on swarm health.

Apart from the Android SDK and the TrustChain Superapp, the main libraries that are used are shown in 5.1. In the following sections we explain the details of implementing the aforementioned features in a top-down overview, starting with the interfaces.

## 5.2. Playlist overview interface

The playlist overview screen, as shown in 5.1 is the screen that is first shown upon starting the MusicDAO. Here the user is presented a list of playlists, loaded from their local TrustChain database. Currently, each Playlist fragment corresponds to exactly one Release block (see 4.2). The playlists are rendered in real-time based on TrustChain data. The playlists are sorted on their torrent swarm health in ascending order.

## 5.3. Playlist fragment interface

The playlist fragment shows its list of tracks and other metadata, such as the title and artists of the Release (see 5.2). For each track the file size is displayed, and a loading indicator on the right side. This shows, in real time, how much of the track is downloaded. In the example of 5.2, the first track is fully loaded.

## 5.4. Wallet interface

Each device participating in the MusicCommunity is given a private/public wallet identity upon installation of the MusicDAO. The wallet interface (fig. 5.4) shows synchronization status of the RegTest network (see 5.9). Once the wallet is fully synchronized with the blockchain, the private key and balance are displayed as shown in fig. 5.5.

Upon browsing a playlist, a donation button is displayed as shown in fig. 5.6. When pressing this button, the user can select an amount and make a direct donation to the artist or band, in the form of a bitcoin transaction from their wallet. The user enters a value in USD and the corresponding amount in bitcoin is calculated and shown when this field is edited. This is implemented using the XChange library[3] connecting to the Binance trading platform[4]. After confirmation, the transaction is registered on the RegTest network (see 5.9).

[3] `https://github.com/knowm/XChange/releases/tag/xchange-5.0.1`
[4] `https://www.binance.com/en/trade/BTC_USDT`

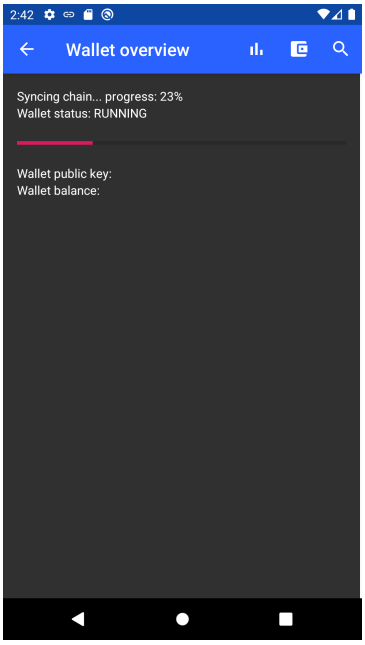


Figure 5.4: Synchronizing with the Bitcoin RegTest environment blockchain

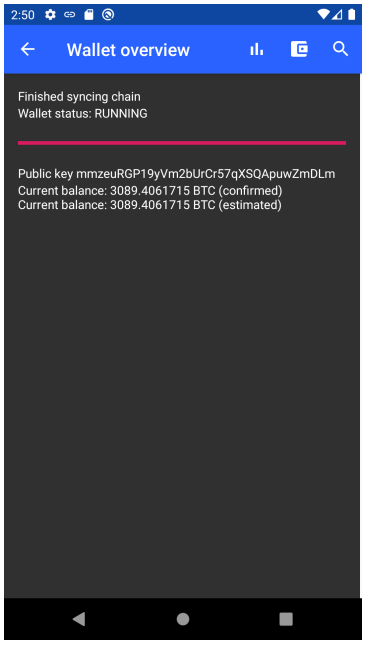


Figure 5.5: Wallet overview and balance after synchronizing

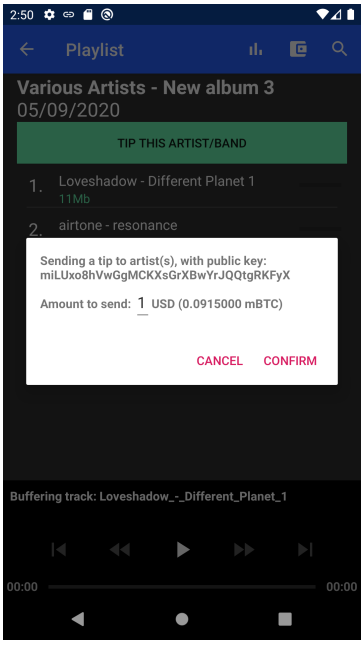


Figure 5.6: Sending a tip to an artist or band

## 5.5. Networking

We implement audio track uploading, downloading and streaming using JLibtorrent[5], an implementation of BitTorrent. Immutable and public storage of metadata is implemented with TrustChain (Otte et al., 2020). This technology has shown to run well on mobile devices (cite), and allows for extensive configuration.

### 5.5.1. Release creation and sharing

To create and share music content, a user can create a Release object using the dialog shown in fig. 5.7. There are two options for creation: the user either selects local audio tracks or pastes a magnet link. Afterwards, the user adds metadata describing the contents and submits the Release. In the background this creates a torrent file, which is stored on the mobile device. Finally, by using the computed infohash and file list, the magnet link of the torrent file is created and added to the *magnet* field of the Release block.

To keep the system independent on centralized trackers, torrent peers are found using a distributed hash table (*Distributed Hash Table*, 1981-2019). In addition, the app uses the local peer discovery (LPD) (8472, 2015) functionality of BitTorrent. This allows for finding peers and transmitting torrent pieces over a LAN, resulting in fast transmission and low latency.

### 5.5.2. Content seeding

Seeding of content is implemented using a simple continuous mechanism. The ContentSeeder class (see X) runs a background thread which seeds all local torrents, with an upper threshold $T$. This is set to $T = 10$. The ContentSeeder uses a last-in-first-out heuristic: Only the top $T$ most recently created/received torrent files are seeded.

## 5.6. Caching

Caching is used for fast browsing and music playback. The process of browsing playlists, loading metadata and selecting tracks using caching is shown in fig. 5.9. All torrent metadata and all track files received from peers are cached on the Android phone. Audio files are stored in the app cache directory[6], indexed by the torrent *name* (Cohen, 2008) property.

[5] `https://github.com/frostwire/frostwire-jlibtorrent`
[6] `https://developer.android.com/training/data-storage`

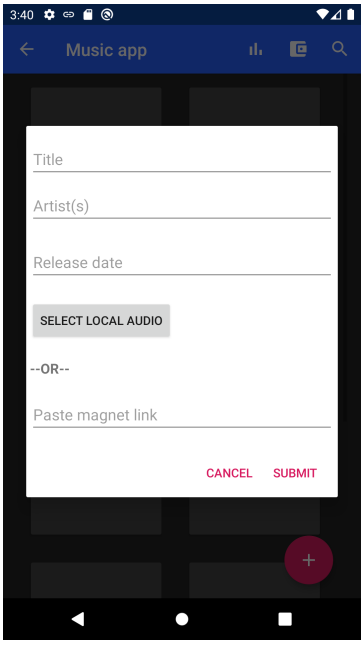


Figure 5.7: Dialog for creating and publishing a new Release

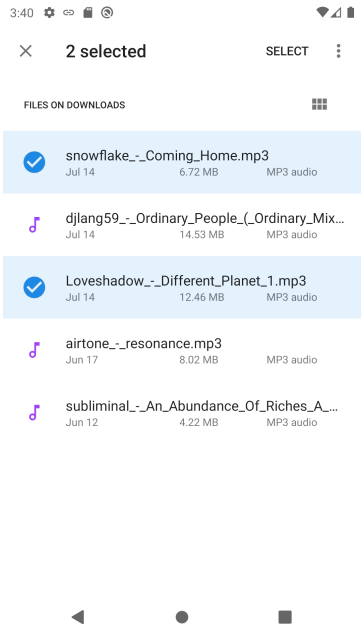


Figure 5.8: Selecting local tracks for creating a new Release

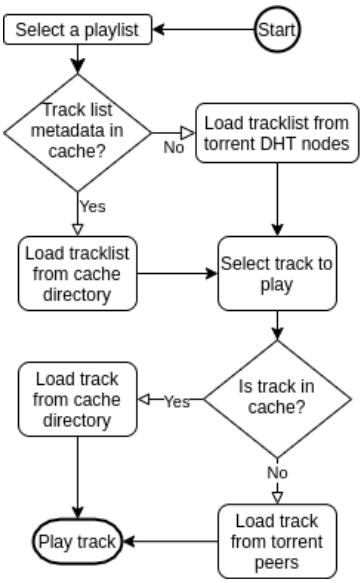


Figure 5.9: Browsing and playing tracks

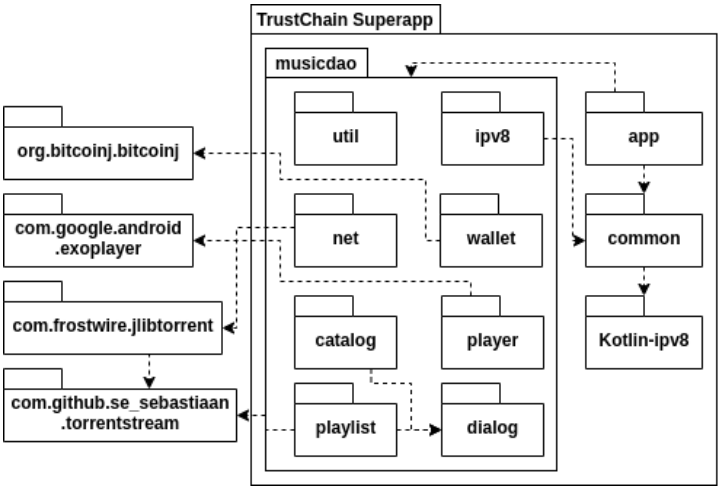


Figure 5.10: Package diagram showing the interaction with external libraries and TrustChain Superapp packages

## 5.7. Music Player and Streaming

Playing music is implemented using ExoPlayer (**??**). This music player library is suitable as it allows for playing tracks that are partially loaded, which enables streaming.

### 5.7.1. Priority handling

To achieve fast buffering, we implemented a priority system for tracks and torrent chunks. This uses the piece priorities system in libtorrent, which range from 1 (normal) to 7 (highest) (see libtorrent Manual[7]). By default, files have a priority of 1. When a user selects a track, this track is given a *file_priority* of 5. In addition, the first 5 pieces of the selected track are given a *piece_priority* of 7, so that the first seconds of the track are buffered quickly and the user can start streaming early.

When the user seeks through a track, the corresponding pieces of the track are given a high priority. Once 2 seconds after the cursor are loaded, the music player starts playing the track. This way the user can start playing the portion of the track they are most interested in quickly.

## 5.8. Identity and authenticity

To identify artists and test the authenticity of Release blocks, we implement the public key infrastructure as described in 4.3. We use the identity system proposed by Skala (2020). All Release blocks are assigned an identity of the creator using a public key. Every device running the MusicDAO will receive a public/private key-pair upon first launch. Currently there is no multi-signature support implemented. This means that in the case of a group publishing a Release, there is only one public key representing the whole group. Every artist, and every unique collaboration between artists should generate their own key-pair to describe ownership of the Release.

### 5.8.1. Keyword search

The app allows users to search for music content remotely using keyword search. This is done using the methods *performRemoteKeywordSearch* and *onKeywordSearch*. These methods test each *title*, *artists* and *date* field of each Release block against a `contain(keyword)` filter. If the keyword is contained in one of those values, it is added to the results.

When the user performs a search, the local database is filtered first to find matches. If there are less than $x$ results, the device asks other peers, with a maximum of $P$. This asks neighbours to inspect their local database to find matches for the same query. If the peer finds a match, it sends the corresponding TrustChain block directly back to the original asker. To disallow packages from endlessly being forwarded on the network we use a time-to-live property (see 4.3).

## 5.9. Donations and payments

We created a public Bitcoin RegTest environment[8] to test peer-to-peer Bitcoin donations and payments. This creates a new 'clean slate' Bitcoin blockchain and allows for full control over the chain and miners. This enables a test environment that is useful for experiments, as we can tweak the block generation speed and keep track of all transactions registered on the closed-off blockchain.

## 5.10. Quality assurance

In order to preserve quality of code we make use of continuous integration, automated tests and an online crash reporter. Unit tests are written using JUnit 4[9] and the mocking library Mockk[10]. The code coverage of these unit tests can be seen in 5.2. The majority of uncovered code contain user interface interaction and networking callback logic.

These tests are run in an automated testing environment using Github Actions[11]. This continuous integration system is executed on every pull request for changes to the code. Aside from this, during the experimentation phase, app installs and crashes are recorded using Firebase Crashlytics, which is a crash reporter

[7] `https://www.libtorrent.org/manual-ref.html#file-format`
[8] `https://developer.bitcoin.org/examples/testing.html#regtest-mode`
[9] `https://junit.org/junit4/`
[10] `https://mockk.io/`
[11] `https://github.com/features/actions`

| Package/class | Line coverage |
|---|---|
| MusicService.kt | 15% (15/95) |
| MusicBaseFragment.kt | 50% (1/2) |
| catalog | 24% (35/142) |
| dialog | 20% (24/130) |
| ipv8 | 81% (57/70) |
| net | 90% (27/30) |
| player | 0% (0/50) |
| playlist | 9% (19/203) |
| util | 60% (42/70) |
| wallet | 0% (0/100) |

Table 5.2: Code coverage overview

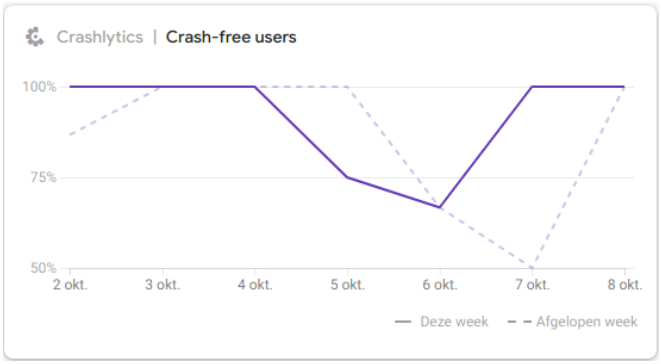


Figure 5.11: Firebase Crashlytics crash reporter. This figure shows the crashes over time peer week

showing details of each application crash of all users (see fig. 5.11). Using this, a full stack trace can be inspected remotely for each crash.

# 6

# Conclusion

# Bibliography


8472, T. (2015). *Bittorrent local service discovery.* Retrieved September 14 2020, from http://www.bittorrent.org/beps/bep_0014.html

aCooke, C. (2018). *Dissecting the digital dollar.* London.

Aguiar, L., & Waldfogel, J. (2018). *Platforms, promotion, and product discovery: Evidence from spotify playlists* (Tech. Rep.). National Bureau of Economic Research.

Andersson Schwarz, J. (2016). Mastering one's domain: some key principles of platform capitalism.

Bucher, T. (2018). *If... then: Algorithmic power and politics.* Oxford University Press.

Cohen, B. (2008). *The bittorrent protocol specification.* Retrieved September 19 2020, from https://www.bittorrent.org/beps/bep_0003.html

de Vos, M., & Pouwelse, J. (2018). A blockchain-based micro-economy of bandwidth tokens. *CompSys 2018.*

*Distributed hash table.* (1981-2019). Retrieved September 14 2020, from https://encyclopedia2.thefreedictionary.com/Distributed+hash+table

Ellis-Petersen, H. (2014). Amazon and hachette end dispute on ebooks. *The Guardian.* Retrieved from https://www.theguardian.com/books/2014/nov/13/amazon-hachette-end-dispute-ebooks

Gillespie, T. (2014). The relevance of algorithms. *Media technologies: Essays on communication, materiality, and society*, *167*(2014), 167.

Haase, C. (2019). Google i/o 2019: Empowering developers to build the best experiences on android + play. Retrieved September 22 2020, from https://android-developers.googleblog.com/2019/05/google-io-2019-empowering-developers-to-build-experiences-on-Android-Play.html

Harvey, D. (2002). The art of rent: globalisation, monopoly and the commodification of culture. *Socialist register*, *38*.

Huang, A. (2019). Super app or super disruption? Retrieved September 21 2020, from https://home.kpmg/xx/en/home/insights/2019/06/super-app-or-super-disruption.html

IFPI. (2018). *Global music report 2018.* International Federation of the Phonographic Industry London.

IFPI. (2020). *Ifpi global music report 2020 - the industry in 2019.* International Federation of the Phonographic Industry London.

Ingham, T. (2018). The odds of an artist becoming a "top tier" earner on spotify today? less than 1%. *Music Business Worldwide*, *25*.

Innis, H. A. (2007). *Empire and communications.* Rowman & Littlefield.

Mulligan, M. (2020). Music subscriber market shares q1 2020. Retrieved from https://www.midiaresearch.com/blog/music-subscriber-market-shares-q1-2020

Otte, P., de Vos, M., & Pouwelse, J. (2020). Trustchain: A sybil-resistant scalable blockchain. *Future Generation Computer Systems*, *107*, 770–780.

Prey, R. (2020). Locating power in platformization: Music streaming playlists and curatorial power. *Social Media+ Society*, *6*(2), 2056305120933291.

ReCode. (2015). *Here's what happens to your $10 after you pay for a month of apple music.*

Skala, M. (2020). *Technology stack for decentralized mobile services* (Unpublished master's thesis).

Srnicek, N. (2017). *Platform capitalism.* John Wiley & Sons.