IOTA-MSS: A Pay-per-Play Music Streaming System based on IOTA

Daniel Melero Martinez
EEMCS
University of Twente
Enschede, Netherlands
d.meleromartinez@student.utwente.nl

***Abstract*—Over the past few years, music streaming has become the main way for users to listen to their favorite songs. Music Streaming Services (MSS) have become the most influential companies over the music industry, with Spotify reaching 406 million monthly active users as of the end of 2021. Unfortunately many artists argue that MSS have created a system which do not fairly reward their music. To address the matter, this paper proposes the implementation of a Pay-per-Play Music Streaming System called IOTA-MSS using the IOTA distributed ledger technology (DLT). This technology provides a secure and scalable platform to perform micro-transactions. IOTA-MSS takes advantage of these characteristics to create a platform in which users can play and distribute music synchronously with the corresponding payments set by its right holders.**

I. Introduction

Since 2017, Music Streaming Services (MSS) have become the foremost contributor to the global recorded music industry's revenue. In the coming years, the number of MSS is expected to continue increasing due to the involvement of large technology companies while the number of physical sales keep decreasing [1].

The best example of a music streaming platform is Spotify, which currently leads the music industry due to its large amount of monthly active users which reached 406 million at the end of 2021 [2]. As opposed to its competition (Apple, Google and Amazon), Spotify only focuses on audio-streaming. Part of its revenue comes from advertising but the large majority comes from premium subscriptions. The way Spotify rewards the right holders of the music in their platform is based on a service-centric model where "an artist's royalty is calculated by taking the number of his or her Spotify streams divided by the total number of Spotify streams. Then, seventy percent of the revenue is given to the rights holder (often a record label or publisher), based on the artist's own royalty rate" [3]. Many argue that this model do not adequately serve the musician or their audiences, these critics are further explored in the following subsections.

A. User's privacy

Most MSS also contain free services to attract new users and hopefully convert them into paying subscribers. As MSS user-bases grow their operational costs do too and unfortunately many platforms have difficulties increasing the number of paying users which produces the majority of their income. To stay competitive "streaming platforms have increasingly refashioned themselves as enterprises whose business extends beyond music-related services to encompass the collection, aggregation, and exchange of user data" [4]. This problem can be seen in Spotify, the largest music streaming platform, which in 2015, changed their privacy policies to allow their application to collect a variety of personal information from their users' devices. This example shows that the current service-centric system used by most MSS may not be adequate enough to maintain the privacy that music audiences require.

B. Musician's compensation

Some argue that the current system do not benefit music right owners either, as it risks compromising a fair environment for the music industry due to "the nature of the royalty arrangements with the streaming platforms, the role of playlists, access to critical data, and the strong negotiating power of the major labels" [5]. This anti-competitive behaviour which is currently in place tends to put at a disadvantage independent musicians which do not have the resources offered by major labels. In most MSS, the main ways for musicians to increase their revenue are mostly out of reach. They can try to get a better contract with the right holders of their music, which should technically be achievable by themselves. But the other existing ways: increasing the overall revenue of MSS or convincing the MSS to share a larger portion of their revenues with right holders, are not directly in their power specially for independent musicians [6]. This is a good example of how musicians are not in control of how their music is being distributed and monetized.

C. User's inability to support musicians

The lack of a sense of control can also be said for the users of these platforms as they are required to pay a monthly fee which do not correlate with their use of the platform. "In the present service-centric system, even if that user never played a single recording by superstars such as Drake or Taylor Swift, a proportion of that user's subscription fee would go to the owners of rights to those superstars' recordings and underlying compositions" [6]. This means that even if musicians have an established audience willing to pay for their music, they might still be unable to support themselves only through their published material.

D. Research Questions

Clearly the current state of MSS do not seem to adequately benefit the producer nor the consumer of the music industry. The right holder of the music has no say on the value of their product and the user cannot control who will benefit from their monthly subscription. It can be argued that to improve the competitive environment of the music industry "creations should be rewarded by either their mass appeal or their ability to command a higher price from the consumer" [3]. This is a known consequence of other models as the one based on voluntary payments where it was concluded that users are willing to pay a fair price for the music they enjoy based on "retailer's personality (values and culture) and the level of reciprocity in the retailer-consumer interaction" [7].

A decentralised solution is deemed to be necessary as the urge to generate profits for shareholders, which seem to be the origin of the mentioned critics, cannot affect the way the platform runs. This paper proposes a novel decentralised music streaming application based on a pay-per-play model. The solution is implemented using the distributed ledger IOTA which provides a secure and scalable platform to perform micro-transactions. The solution allows music right holders to decide the value of their music and users to only pay for the music they listen to. The goal of this paper is to provide a thorough response to the following question:

How can IOTA benefit both the supply and demand sides of the music industry?

To reach a detailed conclusion the paper first responds to the following questions:

- *How can IOTA smart contracts be used to perform rapid high-throughput transactions?*
- *How can data be transmitted in parallel with its corresponding payment?*
- *How can IOTA be used to ensure user's privacy?*

The structure of this paper is as follows, Section II describes the methodology used to answer the research questions. Section III covers a literature review on proposed solutions to similar problems. Section IV provides a detailed explanation of the solution proposed by this paper and Section V collects the results of the analyses ran on the implementation. Finally, Section VI concludes the paper and discusses some ideas for future research to improve the proposed solution's security and reliability.

II. Methodology

To provide a thorough answer to the previously established research questions, a literature review was first conducted. This step provides a wide understanding on the different solutions used by academic literature to address similar questions. An overview of the IOTA white papers was then performed to be able to understand the inner workings of the tangle [8] as well as the IOTA Smart Contract Protocol [9]. The main takeaways from this research were used to implement a working prototype which could perform the basic functionality required. Using the implementation, an analysis of the general performance of the proposed solution was conducted. The results of which are used to answer the different research questions.

III. Related Works

There exist a few proposed decentralized applications aimed at music streaming in the literature. However none of these existing solutions uses the IOTA distributed ledger.

The most relevant work is [10] which proposes a pay-per-play music application based on IOTA. This application uses the Ethereum blockchain to manage the payment transactions and the IPFS protocol to distribute the music. The blockchain is only used to keep track of transactions as well as to publish smart contracts that manages the different intended recipients. In the model described by this paper, the user is able to listen to the songs for free with the option of tipping the artist, for each new block mined in the blockchain 25% of the earnings are divided between the artists in a per-stream basis and the rest is rewarded to the miners. Finally, the songs are stored in IPFS nodes that would be maintained by the artists themselves. This project takes advantage of the Ethereum Smart Contracts to create a system in which music right owners can be rewarded, almost in real time, based on the amount of streams that they receive. Unfortunately, the disadvantage of using Etheureum is that the platform is not scalable and therefore could not replace the current music streaming platforms. Also because the users do not have to pay directly for each stream it means that the current click-fraud that affects major streaming platforms [3] would also be detrimental to the proposed system.

Even though there is no proposed solution for music streaming based on IOTA, there exist some research that proposes data marketplaces based on IOTA. In a data marketplace there are two types of users, IoT devices' owners which have access to a large amount of sensor data and users who are interested in the sensors' data. These marketplaces do not require to stream the data but are still comparable to music streaming platforms as the data transferred could also be audio files.

First, using IOTA 2.0 Smart Contracts, a decentralized marketplace to trade energy in interconnected micro grids is proposed in [11]. In this project an IOTA Smart Contract (ISC) is implemented to allow users to perform uniform double-auctions with their excess energy as the product. The project's setup consists of a separate wasp chain holding the main ISC for each micro grid. both the seller and the buyer are allowed to place bids on the price of energy for their micro grid at each time slot. Finally the same ISC implementation is tested off-ledger in both IOTA and Ethereum and it is determined that IOTA is better suited for the platform as the cost to interact with the ISC is fixed and considerably lower than the Ethereum gas fees. Also thanks to the dynamic Proof of Work used in IOTA 2.0 each transaction consumes much less energy than transactions in the Ethereum blockchain.

Also using IOTA 2.0 Smart Contracts, [12] documents the potential implementation of a data marketplace based on the IOTA Smart Contract Protocol (ISCP) which main focus is

on preserving the privacy of its users. To achieve privacy, the use of a decentralized data storage like IPFS or Swarm to store the encrypted data sold in the marketplace is proposed as a medium that do not require buyers and sellers to interact directly. To achieve security, the use of a distributed certificate authority is necessary to ensure data authenticity. Finally the application's logic would be implemented into three different L2 chains, each holding a different Smart Contract. There would be the buyers' SC and the sellers' SC which would run in a public permissionless environment and would allow buyers and sellers to interact with the platform. The third SC is called the broker and it is an intermediary between the buyers' SC and the sellers' SC. The broker would run in a permissioned environment which is technically not a decentralized solution but it would allow to separate events concerning buyers and sellers making the task of tracing the different data deals really difficult and therefore ensuring users' privacy.

IOTA 1.5 is used in [13] to create a new protocol called Streaming Data Payment Protocol (SDPP). This paper documents the design of this application layer protocol which "enables a buyer and seller to easily connect and transact with each other using micropayments for streaming IoT data". This protocol uses three different channels of communication. The application layer of a standard TCP connection is used to transmit the sold data. Also, IOTA value transfers are used to perform the multiple transactions which covers the payment channel as well as the record medium. Finally the paper offers an implementation of the protocol which is determined to be quite slow for streaming due to the IOTA tangle not being mature enough technology as of 2018. This project uses IOTA 1.0 and therefore do not have access to Smart Contracts as other solutions do in the literature. Instead the researches use Masked Authenticated Messaging (MAM) to implement the record medium.

The same research team that produced the previous protocol, then used it in [14] to develop a Decentralized Data Marketplace. This research paper documents its development and provides the source code of the decentralized application. In this project, IOTA is only used as the payment channel. In contrast with the protocol's first implementation, IPFS and Ethereum Smart Contracts are used to store the different product offers and to keep a record of each transaction. The final result of this research is a working prototype in the form of a web application. Just like the last research paper, this project also uses IOTA 1.0 and therefore do not have access to Smart Contracts in IOTA. To compensate for this, The Ethereum blockchain is used to run the main smart contract which serves as a data structure to keep track of the existing product as well as offering the functionality of adding new products. The downside of using the Ethereum blockchain is that there are fees with each new interaction with the smart contract. It is therefore consider quite expensive to store information on-chain that is why they have to use a distributed file storage to store most of the product information increasing the processing overhead.

In [15] a P2P file-sharing system is created by integrating BitTorrent with IOTA. The protocol described in this paper is able to perform basic BitTorrent functions on an IOTA distributed ledger. The protocol allows users to publish, seed and download files in an environment of higher security inherited by IOTA's advantages. The paper also shows that the protocol reduces the number of unnecessary transaction searches making it faster for users to seed files and produce heartbeats. This project is also done entirely using IOTA 1.0, which means that the main functionalities are performed by personalized IOTA nodes and MAM transactions. It only focus on facilitating the main BitTorrent which means that the tangle is used to keep track of the files being shared as well as its seeders but do not takes advantage of IOTA's micro transactions. The researchers do mention in the conclusion their plan of using IOTA 2.0 smart contracts to transform their system into a P2P file trading system instead. This idea would make sense as smart contracts would allow the BitTorrent functionalities to be performed directly in the IOTA network instead of by a single node.

There also exist many other proposed decentralized marketplaces that are not based on IOTA. The most relevant is [16]. In this paper the use of different blockchains is discussed. IOTA is decided to not be adequate "due to centralization concerns and persistent storage needs" which were problems that the Distributed Ledger had in 2018, the date of publication of this paper. Also the researchers of this paper aim for an "always-on" marketplace which requires the application logic to run directly in the blockchain in the form of a smart contract which IOTA did not have in its first version. It is concluded that a combination of Ethereum Smart Contracts and Swarm, a decentralized storage service similar to torrent, should be used. The paper provides a detailed explanation of the action flow between vendor and customer as well as the Smart Contract's code. The entire application logic is contained in one Smart Contract (SC) and both vendor and customer have to interact with it to perform a trade. The data sold in the marketplace is first encrypted and stored in the Swarm storage system before registering it through the SC. Customers request data to the SC which notifies the vendor. The vendor then sends the decryption key to the customer which then decrypts the requested data with it. This project successfully implement a decentralized marketplace using Ethereum, the researchers mention a few disadvantages of using the Ethereum blockchain. Mainly that block creation times are slow and each transaction requires a small fee which makes it impossible to perform high amounts of micro transactions, the project uses payment channels which is an off chain solution that reduces the latency as well as the fee costs.

IV. Implementation

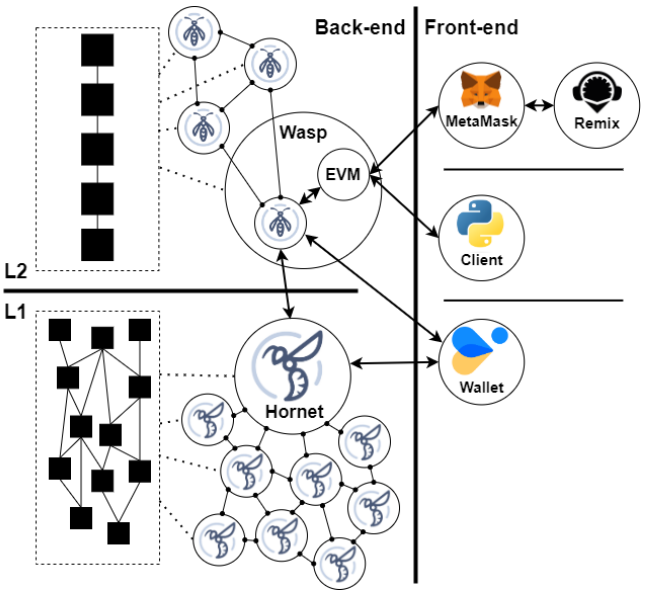
This section will first give an overview in IV-A of the different technologies used by the platform and how these interact between each other. Then, a more detailed explanation of how the user interact with the platform is given in IV-B and how transactions are managed within the smart contract is detailed in IV-C. Finally, the main actions that the users can

execute and how they affect the state of the smart contract are explained in more depth. In IV-D how creating user accounts works, in IV-E how music is uploaded to the platform and in IV-F how sessions are used to listen or download music.

A. Platform Overview

The platform can be divided into two different groups: the back-end and the front-end. The following diagram shows the different technologies which form each group and how these interact with each other.

Figure 1: Diagram of the proposed solution



On the left side of Figure 1 are placed the technologies required to run the smart contract. The Hornet node version 2.0-rc is the software that constitutes the IOTA ledger L1, also called the tangle. The Wasp node version 0.4.0-alpha.2 constitutes the blockchain in which the EVM runs the smart contract, this ledger is also called L2 and is anchored to the previously mentioned ledger L1. Both of these technologies would be running on multiple computers to ensure the ledger consensus but on this implementation only one of each runs on the same local network for testing purposes.

On the right side of Figure 1 are placed the technologies used by different stakeholders of the platform to interact with the smart contract and the two ledgers. First, there is the administrator's front-end which is formed by the tools used to manage the platform. In this group, MetaMask 10.23.3 and Remix IDE Online 0.29.2 are used to compile and deploy the smart contract as well as to perform low level transactions. Secondly, there is the user's front-end which is formed by the python client. This program connects to the EVM using the python library Web3.py 6.0.0b9, establishes TLS encrypted connections with other clients and plays the music it receives using python-vlc 3.0.18121. Finally, the wallet wasp-cli v0.4.0-alpha.2 is used by both administrators and users to set up new chains and transfer funds from the ledger L1 to the ledger L2.

Both front-end and back-end forms the infrastructure behind the proposed music streaming platform, the implementation of which is open source. In https://github.com/DanielMelero/IOTA-MSS, the source code of the python client can be found as well as the smart contract code and instructions on how to set up and deploy the platform on a local network.

B. Platform's Action Flow

All users interact with the platform through the python client which is a command line interface used to perform basic actions intuitively. The following image shows the main menu of the client and therefore all the possible actions.

Figure 2: Screenshot of IOTA-MSS platform menu

All users first have to deposit some funds to their chain address as these are necessary to pay the fees required to perform any action. Then they will create an account with their name and a description. Once the user has an account, the main menu in Figure 2 is available. The user can upload music with option "u" by selecting a mp3 file, entering the name of the song and its price. the uploaded material can be listened or downloaded with option "l" or "d" once it is available on the platform, to do so the user's On-Contract balance must be higher than the stated price which is achieved by transferring the On-Chain funds with option "t". If a song is uploaded or downloaded, the user can then choose to serve it to other users with option "s" which will be rewarded with 10% of the song's price for each full playback. Finally, the user can monitor the activity of the server with option "m" and withdraw the funds generated or unused with option "w".

These individual actions modify the state of the smart contract which simply means that the data of the program is updated acting as a sort of database. The main data structures and how they are used is explained in the next sections. The following code shows the data structure as defined in the smart contract.

```
1  address owner = msg.sender;
2  uint DIST_FEE = 10;
3  mapping(address => User) public users;
4  mapping(bytes32 => Music) public music_map;
5  mapping(bytes32 => Session) public sessions;
6  mapping(bytes32 => uint) distributor_index;
7  bytes32[] public music_list;
8  struct User {
9      bool exists;
10     string username;
11     string description;
12     string server;
13     uint balance;
14     bool is_validator;
15 }
16 struct Music {
17     bool exists;
18     bool is_valid;
19     address author;
20     string name;
21     uint price;
22     uint length;
23     uint duration;
24     bytes32[] chunks;
25     address[] distributors;
26 }
27 struct Session {
28     bool active;
29     address listener;
30     address distributor;
31     bytes32 music;
32     uint price;
33     uint balance;
34     bool[] is_chunk_paid;
35 }
```

Listing 1: Smart Contract's data structure

C. Real-time compensation

A problem that can arise from performing micro-transactions in the blockchain is that the increasing amount of transactions can rapidly reduce the efficiency of the platform as well as increase the gas costs of the rest of transactions. To deal with this potential issue, transactions between users are abstracted within the smart contract in the form of accounting.

Users first have to transfer their MIOTAs, currency of the tangle, to their chain address. During this process, the tokens are transferred to the chain's wallet which holds it and provides the user's address in the chain with the equivalent amount. Then they can deposit these funds into their smart contract account using the payable function `deposit()` which means that the currency sent in the transaction is transferred to the smart contract address which holds it and keep track of the user's balance as a variable.

When a user pays for the music they listen to, a non-monetary transaction is sent to the smart contract which executes the function that takes care of the accounting corresponding to the payments. This process highly reduces the amount of transactions in the system as only one is required and the tokens never leave the smart contract address.

Finally the user can decide to withdraw their account balance using the function `withdraw(uint amount)` which will transfer their account balance to their address in the tangle in MIOTAs. In this process the smart contract sends the tokens to the EVM contract which in turn initiates a transaction from the chain's wallet to the user's address in the tangle with the corresponding MIOTA tokens.

Therefore, the actual transactions with IOTA tokens only occur when the user decides to withdraw their account balance. Users can always choose to withdraw every time their balance increases which would be equivalent to real-time compensation but this do not happen by default to allow for rapid high-throughput exchanges. Further details about user accounts in this solution are detailed in the following section.

D. User creation

The user structure in line 8 is capable of holding all the necessary information about a given user. When a user account is created a new instance of this structure is entered to the users map in line 3, this variable type maps the chain address of the user who sent the request to its corresponding information. Initially the structure only contains the name, the description and exists is set to True, the rest is either set to 0, False or empty string.

The server field can be updated at any point to include the information needed to reach the user's server where the music will be distributed. The balance field is used to keep track of the user's funds in the contract and it is the value that appears in the client menu in Figure 2 as On-Contract balance. It can be increased by transferring funds from the user's On-Chain balance using function `deposit()`, when the uploads are played or by serving music. The balance can also be decreased by withdrawing funds using function `withdraw(uint amount)` or by using the funds to listen to others' music.

Finally the concept of validation behind the last field of the user structure corresponds to the ability to upload music and it is further explained in the following section.

E. Music uploading and distribution

The data structure in line 16 is used to store all the necessary information about a given music file. As done with users, when a music file is uploaded a new instance of this structure is mapped to its id in line 4. This identification value correspond to the keccak256-hash value of the music's name and the address of the owner which allows for multiple files with the same name to be uploaded. The id is then added to the list of files in line 7 so that users know that it is available. The information required to upload a music file is defined by function `upload_music(string _name, uint _price, uint _length, uint _duration, bytes32[] _chunks)`. The author field is set to the user's chain address and the name and price are set by the user itself. The rest of the information is metadata processed by the client using the provided mp3 file. The length corresponds to the amount of bytes and the duration is the music's play time in seconds. The file is divided into chunks to be paid individually and therefore the chunks field corresponds to the list of keccak256-hash values of each chunk. All this metadata is later used by the users to verify the authenticity of the file received.

Once the file is uploaded, it is not directly available to prevent users from sharing music which they do not own the rights which would infringe most countries' copyright

laws based on the World Intellectual Property Organization (WIPO) Copyright Treaty [17]. The responsibility of deciding which files can be shared is, therefore, delegated to the smart contract's deployer. The owner variable is set to its chain address in line 1 and it allows this person to define users as validators using the function `manage_validators(address user)`. files then can be granted or denied validation by these users using the function `manage_validation(bytes32 music)`.

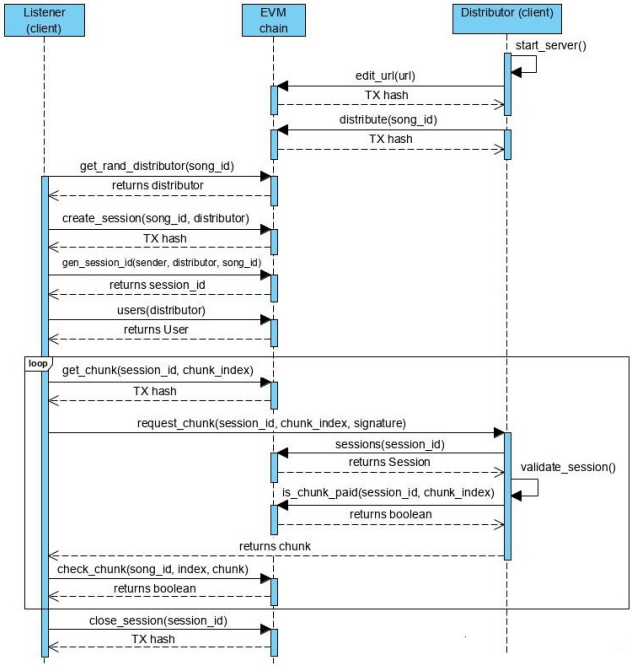
The status "valid" means that the music file can be listened and distributed by others. The author of the music file is set as a distributor because, at first, it is the only one in the platform to have the mp3 file. Other users can also sign up to be distributors using the function `distribute(bytes32 music)` which will add their address to the music's list of distributors.

To distribute a file, users will have to own the mp3 file themselves. They, therefore, require the entire file to be transmitted to them by creating a session and requesting each chunk. This process is further explained in the following section.

F. Session management

Sessions are used to manage the distribution of valid music between two users, a listener and a distributor. This process must be performed efficiently to allow for a decent user experience that is why both users should be using the python client. The sequence diagram in Figure 3 shows an overview of how both clients interact with the back-end and with each other.

Figure 3: Sequence diagram of music streaming



Before any session can be created, at least one user having the mp3 file must sign up to distribute the song. First, the client will start a server and publish the relevant information to its user account using the function `edit_url(string url)`. The url string will contain the server's ip and port as well as the public key certificate used to encrypt all connections. With the server up and running the client signs up for distribution.

On the other side, the listener, having decided on a song, starts the process of creating a session. First, a distributor must be selected, the user might have a preference on which distributor to contact based on its previous experiences, but in case there is no preference the function `get_rand_distributor(bytes32 music)` is available. Then, a session is created using the function `create_session(bytes32 music, address distributor)` which maps the session id with the corresponding session structure. This identification value corresponds to the keccak256-hash value of the listener's and the distributor's address as well as the song's id which allows the listener to create multiple sessions with different distributors for the same music. This can become useful to maintain a song playing even if one distributor disconnects during the session. When the session is created the price of the song gets stored in the data structure in case it is modified while the session is running. Also, the price amount as well as 10% more, the distributor fee, is transferred from the listener's balance to the session's balance. This assures the distributor that the entirety of the song could potentially be paid and, therefore, incentivizes him to maintain an efficient interaction which won't drive away the listener.

Finally, the transmission of the file is a repetitive process in which the listener pays for a chunk of the song and then request it to the distributor. The payment can simply be done using the function `get_chunk(bytes32 session, uint chunk_index)` which transfers, from the session balance, a fraction of the price to the author's balance and a fraction of the distributor fee to the distributor's. Then, the listener establishes a TLS connection to the distributor's server which is encrypted using its public key certificate. The listener sends the chunk request which includes the session id, the index and a signature using its chain address to prove its identity. The distributor then retrieves the session information from the smart contact checks that the index is paid and that the listener's address corresponds with the signature. If everything is correct, the distributor sends the requested chunk back. The listener finally checks that the chunk's keccak256-hash value corresponds to the one uploaded by the author to make sure the data received is authentic. This process is repeated for each chunk necessary to play the song.

When the user finishes or stops the song, the session is closed using the function `close_session(bytes32 session)`. Doing so will return any funds that were not used during the session to the user's account. If the listener never closes the session, the distributor is also allowed to do so because there is a possibility that the chunks will continue to be requested and appear as paid.

This whole process ensures that music can be distributed between users of the platform at the same time that the listener rewards both the author and the distributor. Each connection between clients is encrypted to ensure privacy as well as to defend users from tampering or man-in-the-middle attacks.

V. Analysis

The proposed implementation, mentioned in the previous section, was tested on the local network of a single computer. The back-end consisted of a Hornet node and a Wasp node connected to each other. The front-end consisted of two python programs running in parallel, one taking the role of the distributor and the other the role of the listener. The objective of the following analysis was to measure the two most important metrics of this platform. First, the amount of gas users have to pay to perform basic actions as it is a variable which is not decided by the users themselves but that can affect drastically their experience using the platform. And second, the execution time to perform basic actions as it is also a variable not controlled by users which, if results are excessive, could affect their experience and it might even prevent the platform from offering proper music streaming.

The first metric to measure is the price paid by users to send transactions to the blockchain and interact with the smart contract. The concept of gas is always going to be necessary because nodes protecting the blockchain must be rewarded for the resources they provide. The exact amount can differ a lot from chain to chain, unfortunately , it was impossible to estimate the gas costs of the implementation in a realistic environment due to the development state of IOTA. The Web3.py library function to estimate gas costs, as of January 2023, is not compatible with the EVM implemented by IOTA as it does not yet have a way to retrieve historical gas information. MetaMask is able to estimate the costs without this information but the results cannot be considered a realistic estimation as they are based on the Ethereum blockchain [18]. As IOTA continues to develop the Wasp node, ways to properly calculate gas-based fees are being tested on their networks [19]. Therefore, a realistic analysis on the gas costs of this paper's proposed solution cannot be made until a definitive solution is implemented.

The other important metric to measure is the time to execute any basic action. These values must be kept at a minimum to ensure a decent user experience. After analysing different executions times common to all actions, it is clear that functions that require the most attention are the ones that involve interaction with the smart contract. CPU execution times in the front-end are negligible as the python code used do not perform any complex computation. And the times measured for transmission of data between users are also negligible as the testing environment is a local network. Interaction with the back-end can be divided in two groups: calls which are used to get the value of variables in the current smart contract state and transactions which are used to modify the smart contract state. Table I shows the measured times for each type of interation from the moment the client sends the request to the moment it receives an answer.

Interaction	Result
Call	$\sim 0.1s$
Transaction	$\sim 0.9s$

Table I: Execution times of smart contract interaction

It is clear that blockchain transactions are the processes that really increment the execution times. This might not be an issue for basic actions that only require one transaction as is the case for creating an account, transferring funds or uploading a song. But it can become problematic for managing a session due to the large amount of transactions required. To be precise, when a user listens to a full song, it is required to send one transaction to create the session, then as many transactions as the song has chunks, and finally a last one to close the session. The exact amount is therefore determined by the number of chunks that in turn is determined by how much data the client decides to include in each chunk when processing the mp3 file.

Data in KB	Transactions	Result
5	410	$\sim 10min$
15	138	$\sim 6min$
30	70	$\sim 4min$
60	36	$\sim 2min$

Table II: Music file transmission depending on data per chunk

It is important to determine an optimal number of bytes per chunk as it will allow the platform to transmit songs faster as well as allowing the user to only pay for the chunks used. To determine this value a test was conducted in which a 5 minute song, from a 2 MB mp3 file, was uploaded to the platform using multiple different amounts of bytes per chunk. In Table II, the different uploads were then fully transmitted and the number of transactions as well as the complete duration of execution was measured. The objective of this analysis was to find the smallest value allowing for the file to be transmitted faster that the song's duration. The results of this analysis determined that 30 KB per chunk is the value that will allow for the song to be divided into the most chunks while still allowing for the song to be transmitted faster that it can be listened to.

The analysis' results show that the proposed solution is able to offer music streaming and at the same time manage the corresponding payments. The results also show that the encryption of transmitted data to ensure user's privacy do still allow the platform to function efficiently.

On the other hand, many critics can be made of this analysis. Mainly that the testing environment consisting of a local private network do not simulate the solution on a realistic environment. That it why, it is important to state that further analyses must be conducted that more closely simulate reality to reinforce the validity of the proposed solution. Finally, further analyses must also be conducted concerning the measurement of gas cost using future implementations of IOTA.

VI. Conclusion and future work

Using the distributed ledger IOTA it is possible to stream music synchronously with the corresponding payments set by its right holders. Each research question that was raised at the beginning of this paper was successfully answered. First, IOTA

smart contracts can be used to store users' funds and keep a reliable accounting of the different exchanges, therefore, reducing the amount of necessary transactions to allow the platform to run efficiently. Then, data can be transmitted between users in parallel with its corresponding payments by using the smart contract as an intermediary to keep track of the paid chunks. IOTA can also be used to store servers' public key certificates that are in turn used to encrypt TLS connections between users therefore ensuring their privacy.

Finally, the main research question is also answered as the solution proposed by this paper benefits both the musicians and their audiences. Users are able to listen to their favourite songs, only pay for the parts they request and distribute them to others for a fair compensation. It also benefits musicians themselves as they are able to share their music, have full control over its market price and be rewarded in real-time as users enjoy them.

A problem that can be observed in the proposed implementation is that music only becomes available once it has been validated by a user with special permissions. This part of the solution is necessary to avoid copyright infringements but it might constitute a bottleneck in efficiency and a centralised single point of failure. This matter only affects the process of uploading music but it could be crucial to ensure a decent user experience, therefore, future research should be conducted to address the problem.

Another issue that could arise with the proposed solution is the one of privacy. The information of which music each user requests is public on the blockchain. This is not a problem by itself as users' identity is limited to a wallet address which acts as a pseudonym. The problem is that nodes and distributors can read the IP addresses of users that interact with them. The ability to link both values might become an issue as it is well known that music audiences' behaviour is marketable information. Future research should also address this matter.

REFERENCES

[1] R. A. Rahimi and K.-H. Park, "A comparative study of internet architecture and applications of online music streaming services: The impact on the global music industry growth," in *2020 8th International Conference on Information and Communication Technology (ICoICT)*, pp. 1–6, 2020.

[2] M. Johnston, "How Spotify makes money: Premium Service generates the biggest share of revenue." https://www.investopedia.com/articles/investing/120314/spotify-makes-internet-music-make-money.asp, Mar 2022. [Accessed 29-Jan-2023].

[3] J. Dimont, "Royalty inequity: Why music streaming services should switch to a per-subscriber model," *Hastings Law Journal*, vol. 69, pp. 675–700, 02 2018.

[4] E. A. Drott, "Music as a technology of surveillance," *Journal of the Society for American Music*, vol. 12, no. 3, p. 233–267, 2018.

[5] D. Antal, A. Fletcher, and P. Ormosi, "Music streaming: Is it a level playing field?," *CPI Antitrust Chronicle*, 2 2021.

[6] D. Hesmondhalgh, "Is music streaming bad for musicians? problems of evidence and argument," *New Media & Society*, vol. 23, no. 12, pp. 3593–3615, 2021.

[7] T. Regner, "Why consumers pay voluntarily: Evidence from online music," *Journal of Behavioral and Experimental Economics*, vol. 57, pp. 205–214, 2015.

[8] S. Popov, "The tangle," *White paper*, vol. 1, no. 3, p. 30, 2018.

[9] E. Drasutis, "IOTA Smart Contracts." https://files.iota.org/papers/ISC_WP_Nov_10_2021.pdf. [Accessed 29-Jan-2023].

[10] S. Chavan, P. Warke, S. Ghuge, and R. V. Deolekar, "Music streaming application using blockchain," in *2019 6th International Conference on Computing for Sustainable Global Development (INDIACom)*, pp. 1035–1040, 2019.

[11] C. Mullaney, A. Aijaz, N. Sealey, and B. Holden, "Peer-to-peer energy trading meets iota: Toward a scalable, low-cost, and efficient trading system," 2022.

[12] H. Farahani and H. R. Shahriari, "A privacy preserving iot data marketplace using iota smart contracts," 2022.

[13] R. Radhakrishnan and B. Krishnamachari, "Streaming data payment protocol (sdpp) for the internet of things," in *2018 IEEE International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData)*, pp. 1679–1684, 2018.

[14] G. S. Ramachandran, R. Radhakrishnan, and B. Krishnamachari, "Towards a decentralized data marketplace for smart cities," in *2018 IEEE International Smart Cities Conference (ISC2)*, pp. 1–8, 2018.

[15] L.-Y. Hou, T.-Y. Tang, and T.-Y. Liang, "Iota-bt: A p2p file-sharing system based on iota," *Electronics*, vol. 9, no. 10, 2020.

[16] K. R. Özyilmaz, M. Doğan, and A. Yurdakul, "Idmob: Iot data marketplace on blockchain," in *2018 Crypto Valley Conference on Blockchain Technology (CVCBT)*, pp. 11–19, 2018.

[17] "WIPO Copyright Treaty (WCT) — wipo.int." https://www.wipo.int/treaties/en/ip/wct/. [Accessed 29-Jan-2023].

[18] "MetaMask User Guide: Gas." https://metamask.zendesk.com/hc/en-us/articles/4404600179227-User-Guide-Gas. [Accessed 29-Jan-2023].

[19] "IOTA Smart Contracts Release 0.3.0." https://blog.shimmer.network/iota-smart-contracts-release-030/, Sep 2022. [Accessed 29-Jan-2023].