BRNO UNIVERSITY OF TECHNOLOGY
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

FACULTY OF INFORMATION TECHNOLOGY
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

DEPARTMENT OF INTELLIGENT SYSTEMS
ÚSTAV INTELIGENTNÍCH SYSTÉMŮ

GAME-ORIENTED NON-FUNGIBLE TOKENS ON SOLANA BLOCKCHAIN

HERNÍ NON-FUNGIBLE TOKENY NA SOLANA BLOCKCHAINU

MASTER'S THESIS
DIPLOMOVÁ PRÁCE

AUTHOR **Bc. ANDREJ ZAUJEC**
AUTOR PRÁCE

SUPERVISOR **Ing. IVAN HOMOLIAK, Ph.D.**
VEDOUCÍ PRÁCE

BRNO 2022

Abstract

Smart contrats on Ethereum are trying to be utilized in many industries. Many cryptocurrencies emerged in recent years as the alternative for Ethereum due to high fees and longer processing times. Solana comes with new programming model and scalable architecture that enables both, lower fees and less processing time. The programming models between Ethereum and Solana are compared. The aim of these thesis is to create game-oriented non-fungible token and game on the Solana blockchain.

Abstrakt

Inteligentne kontrakty z Etherea su skusane v mnohych odvetiach. Vela kryptomien vystupilo ako konkurencia oproti Ethereu kvoli drahym poplatkom a dlhsiemu spracovaniu transakcii. Solana prisla s novym programovacim model a skalovatelnou architekturov, ktora umoznuje lacnejsie poplatky a rychlejsi cas spracovania. V tejto praci su porovnane programovaciem modely Etherea a Solani. Cielom tejto prace je vytvorit herny NFT token a hru na Solana blockchaine.

Keywords

herne NFT, Ethereum, Solana, programming model

Klíčová slova

herne NFT, Ethereum, Solana, programovaci model

Reference

ZAUJEC, Andrej. *Game-Oriented Non-Fungible Tokens on Solana Blockchain.* Brno, 2022. Master's thesis. Brno University of Technology, Faculty of Information Technology. Supervisor Ing. Ivan Homoliak, Ph.D.

Game-Oriented Non-Fungible Tokens on Solana Blockchain

Declaration

I hereby declare that this Bachelor's thesis was prepared as an original work by the author under the supervision of Mr. Ing. Ivan Homoliak Ph.D. I have listed all the literary sources, publications and other sources, which were used during the preparation of this thesis.

.....
Andrej Zaujec
January 23, 2022

Acknowledgements

I would like to thank my supervisor for his endless support and patient guidance.

Contents

Chapter 1

Introduction

Bitcoin was created as a trustless alternative based on blockchain for the monetary system. In the simplicity, Bitcoin can be described as the distributed ledger that prevents double-spending and selected new transactions are appended due to the mutual agreement of participants.

Four years later, the Ethereum based on the same principles as Bitcoin brought more complex computational logic on the blockchain via the smart contracts. Smart contracts, cryptographic "boxes" that contain value and only unlock it if certain conditions are met, can also be built on top of our platform, with vastly more power than that offered by Bitcoin scripting because of the added powers of Turing-completeness, value-awareness, blockchain-awareness and state[7].

The smart contracts use-cases are still being explored and yet, there is still a lot of space undiscovered. Financial sector and government processes are one of the most interesting areas where smart contracts could bring most value by increasing transparency, which results in better trust. On top of this, the financial crisis revealed that even in financial services, it is not always possible to identify the correct present owner of an asset[19]. Smart contracts enabled the creation of many new tokens. In contrast to financial assets whose values depend on cash flows, tokens derive value by enabling users to conduct economic transactions on the digital platform, making them a hybrid of money and investable assets[12]. The tokens can have multiple purposes. The most common purposes are security and utility. The security purpose has similar meaning as the shares in the stocks. The utility one provides some functionality for the user from the network in exchange for the token.

On the other hand, the non-fungible token (NFT) stands out for its uniqueness in contrast to the ordinary token, which is equal to every other token of the same kind. The main point of these tokens is the ownership of them. Many users of NFTs is mostly using them for profits. The NFT space is still in infancy, there are new projects appearing every day. It is hard to distinguish between good project, which serves some better purpose rather than making the author of the project rich.

Transaction processing time and fees for each transaction are important for the users of platform. The more users on the Ethereum results in more transactions needed to be processed, this leads to the higher transaction fees and more processing time for one transaction. Solana is one of the newest Ethereum competitors. Solana emerged as the alternative for the high transaction fees and processing times. New smart contracts runtime and different architecture design plays in favor for cheaper fees and less transaction processing time than Ethereum.

The aim of this diploma thesis is to develop game-oriented NFT on the Solana blockchain. The thesis has the following structure. Theoretical prerequisites including description of Solana architecture is described in the Chapter 2. The smart contracts, programming models of Ethereum and Solana, tokens and NFTs are discussed in Chapter 3. Both of the mentioned chapters includes the comparison at the end between what Solana and Ethereum can offer.

Chapter 2

Theory

The features that enable higher transactions throughput and lower confirmation times on the Solana comparing to Ethereum will be described in this chapter. Firstly, Proof of History (PoH) is described and how it introduces the concept of time in the Solana blockchain. Then, leader vote via the Proof of Stake (PoS) in context of Solana is described. Next, the PoH and PoS mechanism are compared to the Ethereum Proof of Work.

2.0.1 Blockchain

Satoshi Nakamoto was able to solve the problem of establishing trust in distributed system via the blockchain. More specifically, the problem of creating a distributed storage of timestamped documents where no party can tamper with the content of the data or the timestamps without detection[13]. This distributed storage is named blockchain.

Blockchain is the appendable, immutable, distributed ledger. The highest difference between blockchains is its permissions. Both, the Ethereum and Solana blockchain are permissionless models. That means there is no prior authorization required before participation, so anyone can participate in creating the permissionless blockchain. On the other hand, there is permissioned model, which requires authorization firstly.

Blocks are appended to the blockchain by the mutual agreement of the participants. The process of how the agreement should be done is determined by the consensus mechanism, which participants have to follow. A consensus mechanism is a fault-tolerant mechanism that is used in computer and blockchain systems to achieve the necessary agreement on a single data value or a single state of the network among distributed processes or multiagent systems, such as with cryptocurrencies[14].

2.0.2 Blocks and transactions

Blockchain consists of the blocks that are chained and each block references to the previous one. The first block is called genesis block and has no reference to the previous block.

The data integration of blocks is protected by the hashes. Each block contains the hash of the previous block and the state root hash. The state root hash is commonly Merkle tree root hash. The Merkle tree root hash is created from the leafs node to the root node, where leafs are hashes of transactions IDs. One leaf represents one transaction ID hash. Then, a non-leaf node hash is created from the hashes of its child nodes. The Ethereum blockchains with Merkle root as state hash can be seen on Figure 2.1.

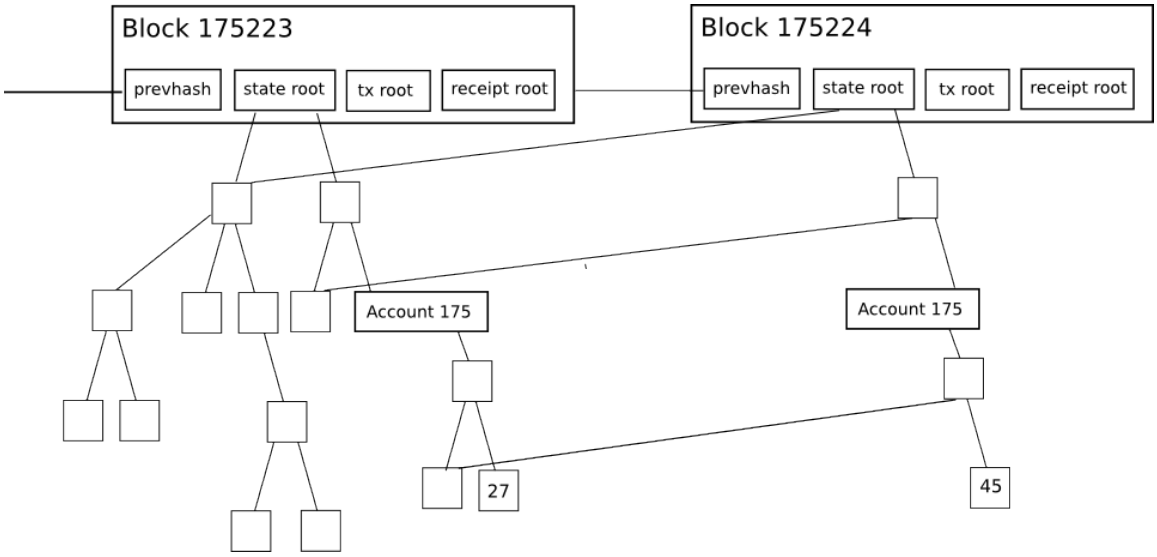

Figure 2.1: Ethereum blockchain[9]

2.1 Solana architecture

Participants of the cryptocurrency cluster are called nodes. Cryptocurrency cluster performance is often measured by the amount of transactions per second (TPS) that the cluster can process. The other important thing to note is the finality of these transactions. This means how long does it take for super majority of nodes to confirm the transaction and make agreement about current state of blockchain.

Both of these mentioned metrics contribute into the block time metric. Block time is the amount of time needed between two blocks in order to append both into the blockchain. The block time does not show which blockchain solution is better, but rather tells which solution has higher transaction throughput and lower confirmation times.

In the time of writing, the block time on the Ethereum is about 13 seconds and the TPS is 16, on the etherscan website[1]. In the meantime, Solana has 615 milliseconds block time and TPS 2,756 on the Solana explorer website[2].

Next will be described which nodes are on the Solana and how each type contributes to the network. Then, Proof of History and Proof of Stake will describe. Following by the process of how is scalable confirmations of transactions achieved.

2.1.1 Nodes

Solana cluster consists of the leader node, validator nodes, lighting nodes.

- leader node – This is the currently chosen node by the Proof of Staking consensus and is responsible for obtaining new transactions from other nodes and pack it into new blocks with new Proof Of History stamp. The newly created block is then propagated to the validator nodes.

[1] https://etherscan.io/
[2] https://explorer.solana.com/

- validator node – Nodes of validator type are securing the network by validating the old and newly created blocks on the blockchain and forwarding newly heard transactions to the leader. The validator node can be chosen as the next leader in the PoS.
- lighting node – Lighting node is a simple user client, which often emit the transactions only regarding the user, which is using this client. These transactions are forwarded directly to the current leader or to the validators node that will redirect it to the leader.

2.1.2 Proof of History

PoH is one of the major features that enables Solana to operate as it is. In the context of Solana, PoH serves as verifiable delay function (VDF).

One iteration of this VDF is hashing its input with the sha256 and yielding the resulting hash and the current iteration number. The verifiable delay lies in repeating that one iteration in the one process, so each iteration is expected to consume some time. Next iteration has the input of previous iteration output, always. The iterations cannot be run in parallel as there is dependency between iterations. The simple sequence of iterations is in Table 2.1.

Iteration	Operation	Output
1	sha256(„random string“)	hash1
2	sha256(hash1)	hash2
	⋮	
200	sha256(hash199)	hash200

Table 2.1: PoH sequence

The other useful feature of PoH is implicit ordering of the results. With the same initial input, we can easily determine in which iteration happened each output, so there is no possible shuffling of the outputs order.

This ordering comes handy, when messages reference some of these outputs. If the message is referencing some output, we will be sure that the creation time of the message is later than the output time. The message happened later in time.

There can also be additional input to the iteration of the VDF. For example, the message with the previous state can be served as input. In that case, we can be sure that message happened before the next iteration and all iterations after that and each yielded output in iterations happened after the message. The PoH iterations with user input can be seen on Figure 2.2.

The validation of the PoH sequence can be scaled horizontally. The horizontal scaling is enabled due to an already known sequence of hashes produced by the leader node, so there is already known input and output which should be achieved. Each validator node divides the iterations into chunks and can spawn one process per chunk. Thus, validation phase is much faster than the generation phase in terms of PoH. Example of divided iterations in horizontal validation is on the Table 2.2.

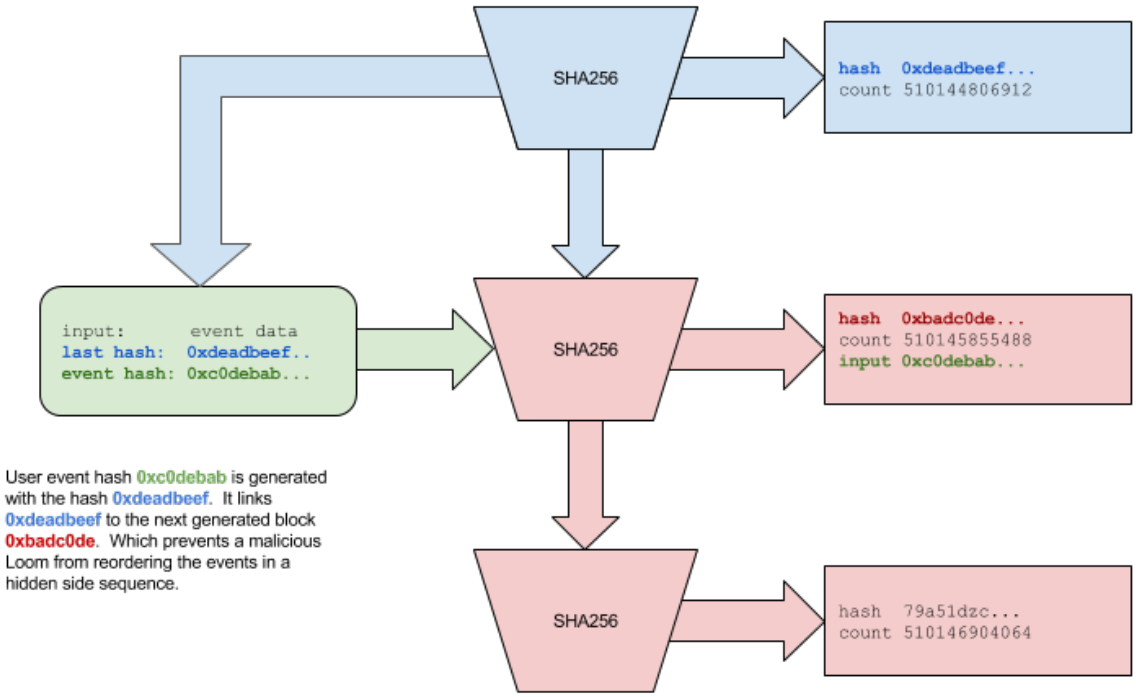

Figure 2.2: PoH with user input[26]

Iteration	Operation	Process
1-99	sha256(„known init string“)	1
100-199	sha256(hash 99.)	2
200-299	sha256(hash 199.)	3

Table 2.2: PoH horizontal validation

2.1.3 Proof of Stake

The Proof of Stake (PoS) in context of Solana is designed for quick confirmation of the sequence produced by the PoH generator that is produced by the current leader. The terminology is explained in the following listing.

- bonds – Bonds are coins from validators that serves as collateral during the validation of transactions. It is the equivalent of the hardware expenses in terms of the Proof Of Work.
- slashing – The proposed solution to the nothing at stake problem the Proof of Stake system[8]. It servers as incentive in order to prevent validators from confirming multiple branches. In case the proof of voting is published, the branch can destroy the validators bond.
- super majority – Is an 2/3 of the validators nodes weighted by their bonds. If the upcoming branch of blockchain has super majority, it will be the sign of established consensus between validators.

It is expected that the PoH generator, which is current leader, will publish a signature of the state before voting period. During the voting, each bonded validator must confirm

the state signature by publishing their own signature of the state. Validators are only able to vote yes. When super majority of validators have voted within specific time window then this branch is accepted as valid.

The election for the new PoH generator are held in case the old PoH generator fails. New leader is chosen as the validator with highest voting power, which is the one with most coins in stake or the one with highest public key address in case of tie.

At any given moment, a cluster expects only one validator to produce ledger entries. By having only one leader at a time and PoH, all validators are able to replay identical copies of the ledger.

2.1.4 Scalable transactions confirmation

Confirmation times are expected to increase only with the logarithm of the number of validators, where the logarithm's base is very high[3].

For example, if the base is one thousand, it means that for the first thousand nodes, confirmation will be the duration of three network hops plus the time it takes the slowest validator of a super majority to vote. For the next million nodes, the confirmation increases only by one network hop.

This fast network convergence is possible thanks to feature named Turbine in Solana. In blockchains, the Scalability Trilemma problem is about optimizing only two of three properties. These properties are decentralization, scalability, security[17]. The Solana looks at the Trilemma problem as a problem with limited bandwidth per peer and tries to optimize that instead.

The Turbine feature stands for dividing the network of validators into the neighborhoods. A stake-weighted selection algorithm constructs the tree such that the higher staked validators are at neighborhoods closer to the leader. Each validator independently computes the same tree[1]. Each neighborhood is responsible for propagating the information into neighborhoods directly below them in the tree.

With all the knowledge, the scalable confirmation can be achieved using the follow combination of techniques.

- Leaders timestamps transactions with a PoH sample and sign the timestamp.
- Leader splits the transactions into batches, send each to a separate node in the closest neighborhood, and have each node share its batch with its peers in neighborhood and neighborhood below him.
- Repeat the previous step recursively until all nodes have all batches and voting begins.
- Voting process utilizes PoS, which is described in Subsection 2.1.3

2.1.5 Transaction fees

Transaction includes the field that says what is the maximum fee. The maximum fee is the upper limit that can be charged by the current leader in order to process the transaction and include it into block. On the other hand, cluster agrees on current minimum fee. The leader will prioritize the higher fee offers, if the network is congested.

Validator estimates the current congestion by counting the signatures per slot (SPS). This indicates how many transactions are currently being processed. The validator node will compare the result with the SPS written in genesis config. The genesis config is setup,

which was used during generating the genesis block. The genesis config states the lamports_per_signature, which is the targeted fee when the network is not congested. The lamports are only small fractions of Solana tokens, this is equivalent to the gwei in terms of fees in Ethereum context.

The client can then ask the cluster for the current fee via the JSON RPC API. The cluster will respond with blockhash and current fee. Each transaction has to contain blockhash in order to prevent duplication and to give lifespan to transaction. So the transaction with old blockhash will be rejected immediately without further processing. The current fee will be valid until the leader rejects the blockhash.

2.2 Summary

This chapter explained what is the blockchain and how blocks of transactions are stored into it. Solana and its major features were described next. These features contribute most to the higher TPS. The Solana TPS is higher than most of the known cryptocurrencies has.

2.2.1 Comparison with Ethereum

Ethereum does use the Proof of Work (PoW) instead of PoS, which is used in Solana. In Ethereum cluster, there is no leader so every participating node is equivalent, and these nodes are called miners.

PoW is more computational intensive than PoS[15]. That is because all miners are simultaneously trying to solve the nonce for the new block. The nonce is a random number that is used just once. The created hash of the block has to meet predefined parameters, and nonce is only part that the miner can change in pursuit of finding the right hash.

Ethereum fees for the transaction are called gas fees. The problem with these fees is their increase and indeterminacy from the users perspective, as the transaction fee can change at the end and the user will have to pay the higher price. To have the transaction executed, the gas price has to be greater than or equal to the lowest Ethereum transaction fees[20]. Miners can filter transaction based on how much fee is each user willing to pay, and they often choose the highest fees for the higher profit. This creates more like biding environment for users. If the user wants to have his transaction confirmed within one or two blocks, the user will have to pay more than other users or just wait more. The height of transaction fees often correlates with the number of transactions that is happening on the cluster and the fiat price of Ethereum as cryptocurrency. This correlation can be seen on the messari metrics[3]. The highest daily average of Ethereum transaction fee was 69,3$ on 11.5.2021, it was the exact date where Ethereum hit one of its all-time highs.

In the times of writing, the Ethereum is currently trying to solve the high transaction fees and minimize the hardware expense and much more. This should be done by transforming from the PoW to the PoS consensus in next years. This transformation already began in 2017[10].

[3] https://messari.io/asset/ethereum/chart/txn-fee-avg

Chapter 3

Programming models of smart contracts

Smart contracts are also called decentralized application(dApps). These dApps are written purposefully to aim the specific features of given programming model, which is used by the cryptocurrency. The programming models in cryptocurrencies can differ a lot, as many cryptocurrencies are trying to make it most suitable for regular users and developers[**?**]. The regular users often see only how many TPS the programming model can bring. On the other hand, developers are interested in how easy is to develop with given programming model or if it is difficult to port already existing dApp to it. The programming model of Solana is described in this chapter. Then, there is the description of the Ethereum programming model, followed by the comparison of these two programming models.

3.1 Smart contracts

The smart contract or dApp, as these names are equal, is run in the decentralized peer-to-peer network. This environment has its benefits but negatives as well. Here are described the most known benefits and drawbacks, which Solana and Ethereum and generally smart contracts have in common.

Pros

- No outage – The network as a whole will always be able to service clients looking to interact with the smart contract once it is launched and on the blockchain. As a result, malicious actors will be unable to perform denial-of-service attacks against individual dApps.
- Privacy – There is no need for providing real-world identity in order to deploy or interact with dApp.
- Resistance to censorship – Users cannot be prevented from submitting transactions, deploying dApps, or accessing data from the blockchain by any single entity on the network.
- Data integrity – Thanks to cryptographic primitives, data saved on the blockchain is immutable and indisputable. Transactions or other data that has already been publicly revealed cannot be forged by malicious actors.

- Verifiable behavior – Without the need to trust a central authority, smart contracts can be analyzed and guaranteed to execute in predictable ways.

Cons

- Maintenance – Because the code and data on the blockchain are difficult to change, dApps can be more difficult to manage. Even if defects or security hazards are discovered in an older version, it's difficult for developers to make modifications to their dApps (or the underlying data held by a dApp) once they've been published.
- Lower computation resources – The program that requires many resources is currently infeasible to run as dApp due to performance bottleneck. Modern computers have huge performance in comparison to dApps sitting on blockchain.
- User experience – It may be more difficult to create user-friendly experiences, since setting up the tool stack required to interact with the blockchain in a properly secure manner may be too difficult for the average end-user.
- Limited runtime – Computation on the Solana and Ethereum blockchain is incentivized. The users who want to execute complex code will be paying high amount of fees.

3.2 Tokens

Smart contracts enabled the creation of many new tokens. In contrast to financial assets whose values depend on cash flows, tokens derive value by enabling users to conduct economic transactions on the digital platform, making them a hybrid of money and investable assets[12]. The tokens can have multiple purposes. The most common purposes are security and utility. The security purpose has similar meaning as the shares in the stocks. The utility one provides some functionality for the user from the network in exchange for the token.

3.2.1 Non-fungible tokens

On the other hand, the non-fungible token (NFT) stands out for its uniqueness in contrast to the fungible token, which is equal to every other token of the same kind. The NFT is one of the kind and mostly aims at its scarcity. The main point of these tokens is the representation of the ownership.

But still, many users of NFTs is mostly using them for profits. The NFT space is still in infancy, there are new projects appearing every day. It is hard to distinguish between good project, which serves some better purpose rather than making the author of the project rich. Many projects that has a lot of market capitalization has power-law distribution according graph-base analysis[11]. This means large entities owns majority of the NFTs in most favorite collections while small users ends up with only few.

3.2.2 NFT properties

As the NFT resides on the blockchain, it inherits many properties implicitly, which would be hard to implement in centralized way. Below are described several

- Tradebility – Every NFT can be traded and exchanded between user.
- Atomicity – Trade can be completed withing one atomic transaction, where each users sees the same state.
- Verifiability – The NFT ownership can be publicly verified as well as history of the owners and prices.
- Transparent execution – Every mint of NFT, as well as every trade, is transparently stored as transactions in blockchain.
- Aviability – The system will always be up and NFT can be traded and minted.
- No censorship – Every user is able to access NFT without blocking only part or specific users.
- Resistance to tampering – The NFT data stored on blockchain cannot be tampered after transaction is settled.

3.2.3 NFT categories

The items traded on the NFT market are grouped into collections, which are groups of NFTs that have some similar characteristics. From packs of collectible cards to choices of art classics to virtual places in online games, collections may take many forms. Art, collectibles, games, metaverse, other, and utility are the six categories that most collections fall into[18]. As the NFT space is still new to the users, there are not set exact borders of the categories. Thus, there is small inter-class distance between the categories so these categories can overlap. Next are described these categories by their utilization in each category. The distribution of each category according market cap can be seen on Figure 3.1 within last four years.

- Art – This is still the most dominant, which holds almost 80% of whole NFT market cap. Any form of digital art can belongs here. This counts for images, drawings, short videos, music samples. Most significant representative is CryptoKitties[1]. This project aims to sell breedable cats where each breed has different scarcity.
- Collectibles – This category represents digital collectible cards inspired by real life collectible cards that are sold at the newspaper stands. Biggest representative is stf.capcom[2] project, which sells collectible cards of street fighters.
- Games – NFT games provide a two-pronged experience for participants, allowing them to gather NFT game items while also competing with their own items. Because participants have a clear purpose in mind: to compete and grow their assets also known as play and earn, NFT games provide some of the most engaging user experiences. The whole game logic cannot be outsourced to the smart contracts as its computationally heavy. Thus, the main game logic is computed off the contracts and only game related items, avatars ownership is done in smart contracts. Most notable project is Alien Worlds[3] where players use the digital items in the alien worlds, players can as well participate in staking in order to increase their chance for higher rewards.

[1] https://www.cryptokitties.co/
[2] https://streetfighter.cards/
[3] https://alienworlds.io/

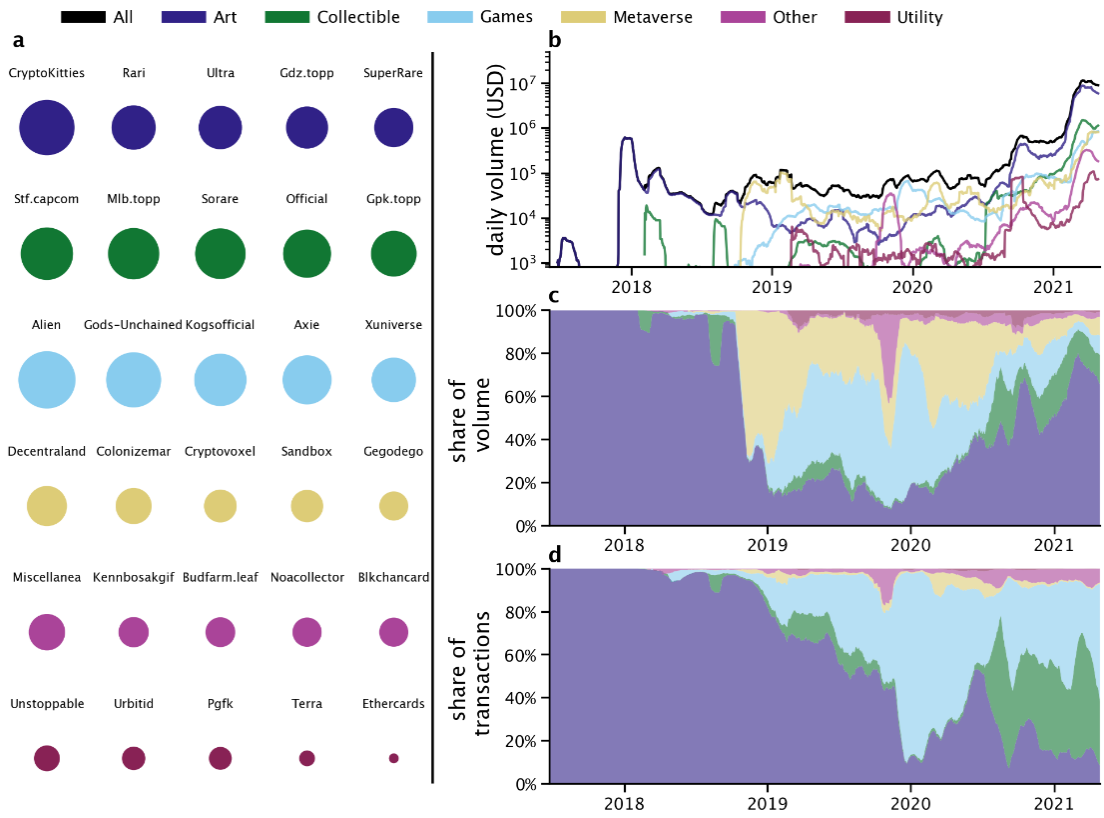

Figure 3.1: NFT categories and their respecting distributions[18]

- Metaverse – Metaverse category could be classified as a subset of games category that detached just recently. Metaverse could be described as the virtual game that simulates an alternative reality where players can use and inspect their bought metaverse items. Virtual lands and estates in metaverse worlds are the most selling items. The biggest project is Decentraland[4].
- Utility – This type of NFT tries to put added value to the scarcity factor. Utility NFT holders could gain access to an exclusive membership, an event, access a game, own the domain. There is still a lot of undiscovered space in this category. The most notable project in this category is Unstoppable[5]. This project sells domains as NFTs.
- Other – This are NFTs of small collections that are not included in the other already mentioned categories.

NFT Mint

The process of creating NFT with the specific smart contract is called mint. This stands for publishing the NFT data to the blockchain and showing the owner of the minted NFT. Minting is typically the last step during creating NFTs.

Art category is the most dominant one from market capitalization as described in 3.2.3. It could be also reason that artistic NFT can be easiest to create for developers because

[4]https://decentraland.org/
[5]https://unstoppabledomains.com/

it does not need anything than picture itself. Most of the art NFTs are created with the help of generative art. The generative art is algorithmically created art only with little help of the developer. The little help is only in providing specific seed for the artwork. This way can developer pre-generate thousands of different pictures and publish the seeds as the blockchain data, which uniquely identify the artwork and users are happy to buy those. Example of generative art project is HashLips[6]. The HashLips engine can generates images of eyes that contains multiple layers that can change, example of layers are background color, eye position, iris color. Each configuration of this layer creates a different scarcity, and some can be more rare than others. In NFT space, rareness directly implies the price.

Other approach for generating art is using General Adversarial Networks (GAN). The authors of this study give people to rate the artworks based on interest, inspiration, innovation and overall impression[23]. The authors used the result to create annotated dataset. This dataset was used to train the known StyleGAN2 architecture and newly created artworks were presented once again to same group of people that rated the first artworks. The results were almost identical, so people did rate the artificially created artwork almost same as the non-artificial one.

Another approach could be to target the specific fans of some already famous art and move to the NFT space. As example, this study created the short GIF as an NFT artwork[16]. The GIF is about bipartite graph representing connection in worlds names mentioned in Isaac Asimov fictional books, thus targeting the Asimov's fans.

To sum it up, creators can make NFT out of anything, but it is another question that it holds some value if any.

NFT Extensibility issues

The main two points in NFT extensibility issues are the interoperability and the updatable NFTs[25].

- Interoperability – Main problem with interoperability is that when user buys some NFT in the given ecosystem, it stays in the given ecosystem and cannot move it or combine it with other interesting projects. User is only let to buy/sell/combine in the given ecosystem. This restriction is mostly due to underlying blockchain technology, which is not designed to communicate with other blockchains. The cross-chain solution can be achieved with trusted third party, but this could result in the loose of decentralization. But most project are still based on the Ethereum, so there can be achieved interoperable at least within platform.

- Updatable NFTs – Another issue is the updating or further existing NFT collection. This is unable due to the immutability of blockchain. Another way that many project use is to store only the main attributes of given NFT as the seed or the unique attribute identifiers and the resulting file is stored on the IPFS file system provided by someone else.

3.3 Solana programming model

When an front-end application communicates with a Solana cluster, it sends transactions containing one or more instructions via RPC API. The Solana runtime forwards those

[6]https://github.com/HashLips/generative-art-node

instructions to dApps that have already been deployed by developers. For example, an instruction may tell a program to transfer Solana tokens from one account to another or to create an contract controlling how Solana or other tokens are exchanged. For each transaction, instructions are performed sequentially and atomically. All account changes in the transaction are discarded if any instruction is invalid. These concepts are explained more detailed next.

3.3.1 Programs and Accounts

Smart contracts in Solana ecosystem are often called programs. Programs and accounts are tightly close, and it is hard to explain only one without the other. That is the reason why terms program and account is coupled in a lot of definitions. Solana separates between code and data. Programs are contracts with no state. There are no class variables, global variable, or anything like that in Ethereum Solidity. On the other hand, data is stored in accounts.

Account types and fields

The account is often called the buffer with extra field between developers, which explains a lot about its data storing property. Each account also has an owner, and only the owner may debit the account and adjust its data. Crediting may be done by anyone. Accounts are owned by program. The default account owner is the native System program.

Accounts are differentiated by their unique addresses, which is public key. All data to the program is sent by reference from the outside, the reference is address of account. There can be different types of the accounts, these types are described. next[5].

- Data accounts that store data can be separated further into.
  - System owned account that store program data
  - Program Derived Address (PDA) accounts that serve for cross-program invocation
- Program accounts store executable code
- Native accounts that store native programs such as System, Vote and Stake

Program code is also data, so the program code is stored in accounts, which are marked as executable. Each account attribute is described in Table 3.1.

Rent is paid for storing the data on the account. If the account contains at least a balance of two years rent, then it will be exempt from paying the rent.

Benefits of separated code and data

Parallelization is one of the reason for this separation of code and data. As the exact data references are already known before the program execution, these programs can be executed in parallel as long as the writable address do not overlap between parallel programs.

Another reason for separate code and data is deterministic behavior with same program but different data. For example, the program for creating new tokens does not have to change at all when a new token is minted. The only changed thing is the address of the minted token, thus the data. So, the same programs can be used with diffe/rent addresses.

Field	Description
key	Public key of account
is_signer	Whether transaction is signed by this account
is_writable	If is account writable
lamports	The lamports in program. Modifiable by program.
data	The data stored in account. Modifiable by program.
owner	Public key of owning program
executable	Whether this account stores program code
rent_epoch	The next epoch of account rent

Table 3.1: Account fields[4]

This cannot be done on Ethereum, as each token there has to implement the same or similar smart contract for creating the new token.

Common examples of this pattern are seen across the Native and SPL Programs.

Native and SPL Programs

Solana comes with a variety of programs that serves as the building blocks for on-chain activities. Native Programs and Solana Program Library (SPL) Programs are the two types of programs available.

Native Programs offer the basic functionality needed to run validators. The System Program, which is responsible for administering new accounts and moving SOL between two parties, is the most well-known of these programs. Other native programs are BPF loader, Vote. The Vote program is necessary for proper working of Solana PoS consensus described in Section 2.1.3.

Variety of on-chain activities, are supported by SPL Programs a variety of on-chain activities, including token creation, trading, and lending, as well as stake pool generation. These are not necessary for correct working of the cluster as Native programs but are important to users. The SPL Token Program can be invoked directly via the CLI, while others like the Associated Token Account Program are usually composed with custom programs.

3.3.2 Program writing and deploying

Programs are most commonly written in the C/C++ or the Rust language. Almost all the examples and native programs are written in Rust, so it is preferred over C/C++. The well-used architecture of Rust programs is described in Table 3.2. Every program has a single entry point where instruction processing takes place and parameters of this entrypoint always includes program_id, array of accounts, instruction_data, which is byte array. The entrypoint function is declared via the entrypoint macro in Rust programs.

File	Description
entrypoint.rs	Entrypoint to the program
lib.rs	Registering modules
processor.rs	Program logic
error.rs	Program-specific errors
instruction.rs	Program API, (de)serializing instruction data
state.rs	Program objects, (de)serializing state

Table 3.2: Rust program architecture[6]

eBPF

The code is compiled into the extended Berkeley Packet Filter (eBPF) instruction set. The eBPF is known from the packet capturing and processing in Linux systems, where it loads direcly into kernel for faster processing time. eBPF is Turing-complete in that it incorporates all of the standard instructions found in basic ISA, including jumps (conditional and unconditional).

The eBPF verifier in the Linux kernel prevents loops and recursion, making it non-Turing-complete. This is necessary to ensure that an eBPF software with an infinite loop does not cause the entire system to hang.

Solana supports loops and recursion, but it still controls CPU utilization by counting the number of eBPF instructions executed at runtime.

Deployment

The program is compiled as a binary ELF shared object that contains eBPF bytecode. Then, binary is uploaded to the Solana cluster and deployed.

Programs, like everything else on Solana, are stored in accounts, but these accounts are tagged as executable and given to the BPF Loader, which is the native program. The program id is the account's address, and it is used to refer to the program in any future calls. Now, the program can be invoked and run by the Solana runtime.

Solana offers multiple versions of BPF Loaders. The most recent of which is the Upgradable BPF Loader, which enables the upgrading of the already deployed programs. The BPF Loader is in charge of managing the program's account and making it accessible to users through the program id.

3.3.3 Transactions and instructions

Instruction is smallest unit in terms of programs. It is atomic otherwise it will be reverted. One instruction represents one call to the program. One or more instructions can be bundled. This bundle is message and the message with the array of signatures creates the transaction. Each message has signature in the signature array. Runtime also verifies that each signature was signed by the private key that matches to the public key in the message's account addresses array at the same index.

The instruction specifies a single program, a subset of the transaction's accounts to be passed to it, and a data byte array to be passed to it. The program interprets the data array and executes the instructions on the specified accounts. The program may return with a success code or an error code. When an error return is received, the transaction is immediately terminated.

Fees

Solana distinct between two type of transaction fees.

One is transaction fee, for propagating transactions and is calculated by number of signatures made in transaction, which needs to be verified. The number of signatures is multiplied by the fee rate that is lamports_per_signature and how this value is determined can be found in Subsection 2.1.3.

Other type of fee is storage fee. This fee is closely bounded to the programs storage and is called rent, and it is already described in Subsection 3.3.1.

The transaction fee is paid by the first writable account that has signed the transaction is writable. It has to be writable account due to ownership rule, only owner of account can deduct its account.

Currently, half of the transaction fee serves as profit for validators. Other half is burned to keep the currency deflationary[24].

3.4 Ethereum programming model

Ethereum smart contract runtime is called Ethereum Virtual Machine (EVM). The biggest difference with Solana is that data and code are not separated but rather lives in the same account. Ethereum smart contracts are programmed in the languages that were specifically designed for this purpose. The languages are called Solidity, Vyper. Vyper language has emerged just recently as an alternative for Solidity, but it has not much support yet. Both of these languages are trying to be have In Ethereum ecosystem, smart contracts are called the contrasts or dApps.

3.4.1 Accounts

There are two types of accounts.

- External account - This is controlled by the public-private key pair
- Contract account - This is controlled by the code that is stored in the account

The address of external account is the public key regarding to private key. On the other hand, the address of the contract account is derived with hashing function from the creator address and number of sent transactions from that account so called nonce[2].

External account

External account is more simple one as it stores very few information comparing to contract account. One of the attributes is balance, how much wei it holds. Weis are the small fractions of Ethereum currency, which are used to fee deduction. Weis are equivalent of lamports in Solana.

Other attribute that external account stores is nonce. The nonce stands as the counter of transaction that given account sent. The nonce is incremented each time the new transaction is send. The nonce seems like redundant information for the use of smart contracts but it has crucial role in validation transactions and preventing its replay. The transaction always includes the nonce of sender. The transaction nonce is compared with the current nonce of sender during runtime to prevent executing the same transaction twice or more. If the nonce of sender specified in the transaction does not match with the current nonce of sender, the transaction will be rejected as some other transactions happened already.

Contract acccount

Storage is the persistent key-value store that maps 256bit words to the 256bit words. Every account has storage, but the external account has storage root set to the null hash. This indicates that there is no storage. Thus, only contract accounts have storage that can be used for storing data. The nonce in contract account has other purpose, as the contract account can not be used to send the transactions. The nonce attribute represents the count of contract creations the contract account has created. Other attribute are hash of the EVM byte code that is stored in the contract account. Another attribute is storage root and balance in weis.

3.4.2 Contracts

Contract in Solidity is similar to class in the object-oriented languages. Contract support polymorphisms and inheritance. In similar way, there is this reference for contract to access itself. Keyword this is just syntactic sugar around contract address so whenever contract calls its function, it is the same process as the execution of function of the different contract but the this keyword passes the same address as the caller has. Contracts are calling other contracts via messages.

Storage, memory, stack

The contract contains persistent storage, which can be modified via the function calls of the contract.

Every variable declared on the contract level is automatically stored in the storage and serves as the global variable that is persistent across different contract function.

Variables declared in functions except constructor are in the stack. EVM is stack-basedd machine, not the registers. Each function invocations create the new stack.

Other accessible non-persistant space for contract is memory. The memory is completely new for each message and is flushed after the function execution ends.

Calling functions of other contract will result in EVM function call that switches context so the caller contract variables and state is inaccessible. Contracts can only read and write to their own storage. This is another difference against Solana as the programs can read anything as long as the proper accounts sign the transactions.

State variables of other contract can be accessed only if the other contract exposed it via the getter function.

3.4.3 Writing contracts and deployment

As already mentioned, the main two languages for writing contracts are Solidity and Vyper.

Solidity

- High-level object oriented language
- Most influenced by C++
- Statically typed
- Supports inheritance, libraries
- Big community support

Vyper

- Python programming language
- Strong typing
- Less functions than Solidity in order to make contracts easier to verify and be more secure
- No inheritance, polymorphism, recursive call
- Faster prototyping thanks to Python

The next theory about smart contracts in Ethereum will be solely focused on Solidity as it is the language with biggest community support and most features. The smart contract code compiled into the EVM bytecode, which is then deployed to the network.

Deployment

The transaction creates a new contract if the target account is not set (the transaction recipient is null). EVM bytecode is assumed as the payload of such a contract creation transaction and is executed. The contract's code is permanently recorded as the output data of this execution. This implies that instead of sending the contract's actual code, you send code that returns the contract's code when it is executed. There is no standard way for uploaded contract to change its code in future.

However, there is known pattern as an proxy contract. The proxy contract which sits at the top level and proxies the contract calls to current version of released user contract. Whenever user wants to update the code, the proxy contract will change its references to the newly released contract.

This is another difference with the Solana, where the proxy contract work is done by Upgradable BPF loader program.

3.4.4 Composability

As the Ethereum does not separate code with data, the already deployed smart contracts cannot be reused for the same function but different accounts, as it was in case of Solana described in 3.3.1. What can be reused is only the implementation code. So at the end, you have to deploy the same contracts twice or more times, but you do not have to write the same logic in source code.

When creating smart contracts, developers often find themselves of writing similar patterns over and over. Solidity language offers reusing the same code or extending it is done via libraries or contract inheritance.

The library serves as the already done implementation of functions. It cannot be inherited from library, as well as library is not deployable and does not include payable functions. The use-case for libraries are safer computational functions. For example, safer arithmetic operations.

The inheritance is the same as in Classes in OOP. The developer can then deploy the contract or extend it for his use-case and deploy.

Standards

There is no official implementation of the smart contracts from the Ethereum that can be used as the basic building block like Solana has SPL described in 3.3.1. The Ethereum community rather proposes standards. These standards aim to specify the exact interface and events, which should contract implement for its interoperability. The confirmed and used standards are called Ethereum Improvement Proposals (EIP). There are two most notable EIPs.

- EIP-20: This[7] standard specifies what contract has to implement in order to create a token.
- EIP-721: This[8] standard specifies what contract has to implement for the NFT.

An example of interface function both of this standard specifies is the transfer function. User must be able to transfer tokens whether it is the fungible or non-fungible.

Community contracts

As the feature scope in Solidity contracts its large thus is the attack vector space on these contracts. Over the years many developers have failed to implement contracts in secure way and this had consequences in losing millions of dollars[22]. Example of security pitfall is integer overflow in contracts.

Developers are often prone to make same errors again so this was main reason for companies or community teams to publish community contracts. These serves as building blocks for other contracts and should be safe to use. OpenZeppelin[9] is one example of the company that is doing security audits of contrasts. This company proposed the secure implementation of the described EIP standards and many other contracts that address repeating patterns in more secure way.

3.4.5 Transactions and messages

The transaction is sent from one account to the other, it might include the payload encoded in binary data and Ether.

Message calls allow contracts to call other contracts or send Ether to non-contract accounts. Message calls contain a source, a target, a data payload, Ether, gas, and return data, much as transactions. Every transaction, in fact, starts with a top-level message call, which might then lead to other message calls.

Events

Solidity events are the equivalents of logs in the EVM runtime that can user subscribe to. Other contracts and contract itself do not see the events. The contracts can only emit them. The typical use-case is the notification whenever something happens that is important for a given user that subscribed to the listening of this particular event.

User can subscribe by topic via the RPC interface on the Ethereum client. These logs are stored in blockchain with associated contract address that emitted them.

[7] https://eips.ethereum.org/EIPS/eip-20
[8] https://eips.ethereum.org/EIPS/eip-721
[9] https://openzeppelin.com/

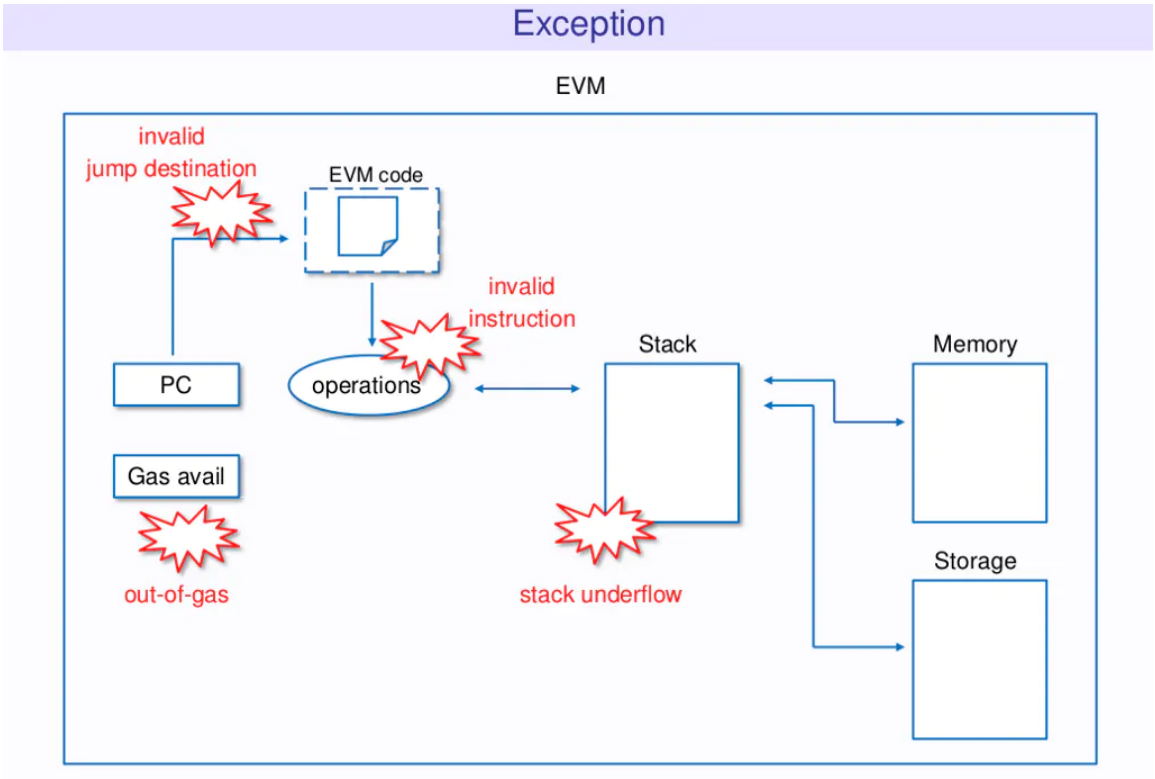

Figure 3.2: Example of exceptions in EVM[21]

Gas

The gas is like pre-paid fee in Ethereum ecosystem that is lowering during execution of transaction. Each transaction is given a particular amount of gas when it is created, with the goal of limiting the amount of work required to complete the transaction while still paying for execution. The gas is steadily reduced while the EVM processes the transaction, according to predefined criteria.

The gas price is determined by the transaction's author, who must pay gas price multiplied by gas from the sending account up front. If any gas remains after the execution, it is returned to the author.

If the gas runs out at any stage, an out-of-gas exception is thrown, reverting any changes done to the state.

Examples of which exceptions can occur during EVM runtime are on Figure 3.2.

3.5 Comparison between Ethereum and Solana model

- Separate code and data – As Solana programs does not hold any state there is less byte code duplicity on the blockchain as the same programs does not have to be redeployed like in Ethereum but rather only changes accounts. This separation of code and data is an overall architecture change and has further implication on the whole programming model. This is described more in detail in 3.3.1.

- Low-level vs High-level language – Solana programs are basically functions whereas Ethereum's Solidity is higher-level language that enables polymorphism and inheri-

tance and has many other language features. This plays favor mostly for Ethereum developers, which can reuse the community contracts 3.4.4 as building blocks and does not have to rewrite same logic twice as Solana developers.

- Scalability – Solana has advantage in higher TPS and faster confirmation times mentioned in 2.1. This higher throughput directly implies the scalability of smart contracts. Thus, users have to wait less and there will not be problem when more users will come.
- Fees – As already mentioned in 2.1, transaction fees are cheaper on Solana. The users can generate more transactions with paying significantly less.
- Ecosystem lockout – This is problem with blockchain solution general already described in 3.2.3. As these blockchains are not able to cross communicate with other blockchains. The user has to choose one ecosystem and be locked there. This is positive for Ethereum due to the majority of projects operates there.
- Community support – Solana is only recent project, so there is not much community of developers there. This results in low amount of developers guide and not many in-depth descriptions in documentation. Many Solana developers rather reference the Solana core code rather than guide because often the guide is missing in specific topic.

Bibliography

[1] Turbine — Solana's Block Propagation Protocol Solves the Scalability Trilemma. *Medium*. 2019. Available at: https://medium.com/solana-labs/turbine-solanas-block-propagation-protocol-solves-the-scalability-trilemma-2ddba46a51db.

[2] Introduction to Smart Contracts. 2021. Available at: https://docs.soliditylang.org/en/latest/introduction-to-smart-contracts.html?highlight=memory#accounts.

[3] A Solana Cluster. *Solana*. 2021. Available at: https://docs.solana.com/cluster/overview#confirming-transactions.

[4] Struct AccountInfo. 2021. Available at: https://docs.rs/solana-program/1.5.0/solana_program/account_info/struct.Accou/ntInfo.html#fields.

[5] AndyNft. Accounts. 2021. Available at: https://solanacookbook.com/core-concepts/accounts.html.

[6] AndyNft. Programs. 2021. Available at: https://solanacookbook.com/core-concepts/programs.html.

[7] Buterin, V. Ethereum White Paper: A Next Generation Smart Contract & Decentralized Application Platform. 2013. Available at: https://github.com/ethereum/wiki/wiki/White-Paper.

[8] Buterin, V. Slasher: A Punitive Proof-of-Stake Algorithm. *Ethereum*. 2014. Available at: https://blog.ethereum.org/2014/01/15/slasher-a-punitive-proof-of-stake-algorithm/.

[9] Buterin, V. Merkling in Ethereum. *Ethereum*. 2015. Available at: https://blog.ethereum.org/2015/11/15/merkling-in-ethereum/.

[10] Buterin, V. and Griffith, V. *Casper the Friendly Finality Gadget*. 2017.

[11] Casale Brunet, S., Ribeca, P., Doyle, P. and Mattavelli, M. *Networks of Ethereum Non-Fungible Tokens: A graph-based analysis of the ERC-721 ecosystem*. 2021.

[12] Cong, L. W., Li, Y. and Wang, N. Tokenomics: Dynamic Adoption and Valuation. *The Review of Financial Studies*. august 2020, vol. 34, no. 3, p. 1105–1155. DOI: 10.1093/rfs/hhaa089. ISSN 0893-9454. Available at: https://doi.org/10.1093/rfs/hhaa089.

[13] Di Pierro, M. What is the blockchain? *Computing in Science & Engineering*. IEEE. 2017, vol. 19, no. 5, p. 92–95.

[14] Frankenfield, J. Consensus Mechanism (Cryptocurrency). *Investopedia*. 2021. Available at: https://www.investopedia.com/terms/c/consensus-mechanism-cryptocurrency.asp.

[15] Gervais, A., Karame, G. O., Wüst, K., Glykantzis, V., Ritzdorf, H. et al. On the security and performance of proof of work blockchains. In: *Proceedings of the 2016 ACM SIGSAC conference on computer and communications security*. 2016, p. 3–16.

[16] Janosov, M. and Borsi, F. *Asimov's Foundation – turning a data story into an NFT artwork*. 2021.

[17] Monte, G. D., Pennino, D. and Pizzonia, M. Scaling blockchains without giving up decentralization and security: a solution to the blockchain scalability trilemma. In: *Proceedings of the 3rd Workshop on Cryptocurrencies and Blockchains for Distributed Systems*. 2020, p. 71–76.

[18] Nadini, M., Alessandretti, L., Giacinto, F. D., Martino, M., Aiello, L. M. et al. Mapping the NFT revolution: market trends, trade networks and visual features. [Scientific Reports 11, 20902 (2021)]. 2021. DOI: 10.1038/s41598-021-00053-8.

[19] Nofer, M., Gomber, P., Hinz, O. and Schiereck, D. Blockchain. *Business & Information Systems Engineering*. Springer. 2017, vol. 59, no. 3, p. 183–187.

[20] Pierro, G. A. and Rocha, H. The influence factors on ethereum transaction fees. In: IEEE. *2019 IEEE/ACM 2nd International Workshop on Emerging Trends in Software Engineering for Blockchain (WETSEB)*. 2019, p. 24–31.

[21] Saini, V. Getting Deep Into EVM: How Ethereum Works Backstage. 2018. Available at: https://hackernoon.com/getting-deep-into-evm-how-ethereum-works-backstage-ac7efa1f0015.

[22] Sayeed, S., Marco Gisbert, H. and Caira, T. Smart Contract: Attacks and Protections. *IEEE Access*. january 2020, PP, p. 1–1. DOI: 10.1109/ACCESS.2020.2970495.

[23] Shahriar, S. and Hayawi, K. *NFTGAN: Non-Fungible Token Art Generation Using Generative Adversarial Networks*. 2021.

[24] Starry, J. Transaction fees. 2021. Available at: https://jstarry.notion.site/Transaction-Fees-f09387e6a8d84287aa16a34ecb58e239.

[25] Wang, Q., Li, R., Wang, Q. and Chen, S. *Non-Fungible Token (NFT): Overview, Evaluation, Opportunities and Challenges*. 2021.

[26] Yakovenko, A. Solana: A new architecture for a high performance blockchain v0. 8.13. *Whitepaper*. 2018.