DeFi Gaming Platform Using the Layer 2 Benefits

Tomas Rafaj, Lukas Mastilak, Kristian Kostal, Ivan Kotuliak
Slovak University of Technology in Bratislava
Bratislava, Slovakia
xrafaj, lukas.mastilak, kristian.kostal, ivan.kotuliak@stuba.sk

***Abstract*—Decentralized finance (DeFi) and blockchain-based gaming platforms are rapidly gaining popularity in the digital world. However, the high gas fees associated with on-chain transactions make it difficult to use these platforms for mass adoption. In this paper, we propose a DeFi gaming platform that leverages the benefits of layer 2 solutions to reduce gas fees and improve user experience. Our platform utilizes state channels, where players can interact with each other without the need for on-chain transactions. Our platform's architecture and smart contracts are designed to ensure security, transparency, and fairness. We demonstrate the feasibility and effectiveness of our platform through several test scenarios. In addition, we implement two approaches to validate the game results for different games. Our results show that using a layer 2 solution significantly reduces gas fees, making DeFi gaming accessible to a wider audience. Our proposed platform can potentially revolutionize the gaming industry by providing a seamless, decentralized, and rewarding gaming experience to users.**

I. Introduction

Blockchain technologies experienced a big rise in the last decade, with many use cases found and implemented on these technologies. The idea of decentralized finance has been and continues to be very successful, with thousands of platforms that implement borrowing, trading, or even copying assets from the traditional finance world. However, decentralized blockchain technology does not have to be limited to the financial sector. The recent rise of non-fungible tokens, known as NFTs, has seen extreme success among people and even found its way into gaming platforms. The need for decentralized gaming platforms still exists, as most projects that have attempted it have either failed or been stopped midway. However, the most commonly used platform is the Ethereum network which provides the best base for smart contracts and the implementation of decentralized applications.

The following sections of this article are organized: We describe components of the Ethereum network and argue why it is suitable for recording and evaluating game results in Section II. Then, Section III shows the current state-of-the-art solutions in the domain of DeFi games. Section IV describes the design architecture of our DeFi gaming platform, which uses a state channel to reduce transaction fees. Evaluation and Discussion to prove our solution is in Section V and VI. Section VII sums up the article with results, and we also discuss future work there.

II. Ethereum blockchain

Ethereum network is also called the Turing machine, which keeps track of all states of everything in the network. The start is called the genesis or genesis block, which is the first block ever made on Ethereum. Its function is to go from the genesis state to the final state, with each new block holding a new final state. Blocks on Ethereum work like a notebook, keeping track of all transactions, the previous block, and the identifier of the final state. However, due to their size, they do not contain the final state themselves. Moreover, they contain data such as transactions, smart contracts, block difficulty, block number, and a link to the latest state [1].

A. Proof of stake

Ethereum blockchain uses the proof of stake mechanism. It means all nodes share a uniform view of the network with the Ethereum peer-to-peer network. They have one global state that they all agree on. Validators are chosen to create new blocks based on their stake, or the amount of cryptocurrency they have deposited as collateral to participate in the network. When a validator is selected to create a new block, they must first verify and validate the transactions in the block. This involves checking that each transaction is valid, that the sender has enough funds to make the transaction, and that there are no attempts at double-spending or other forms of fraud. If a validator behaves maliciously or attempts to double-spend cryptocurrency, their stake can be "slashed" as a penalty. Once a block is confined as valid, the state of the network is changed [2]. The average time between blocks is 15 seconds [3].

B. Blocks

Blocks are a product of the Ethereum network proof of stake consensus. Block contains information, such as block number, hash of the block, parent hash, timestamp, and many others, which we can see in Fig. 1. Block number defines the block rank in the network. Further, the hash of the block is a 32-bit hash code of the block, and the parent hash is the hash of the block above it in a tree structure. Timestamp is the record of the time when a block was produced, and the nonce is the number of confirmed transactions from a given account sent previously [4].

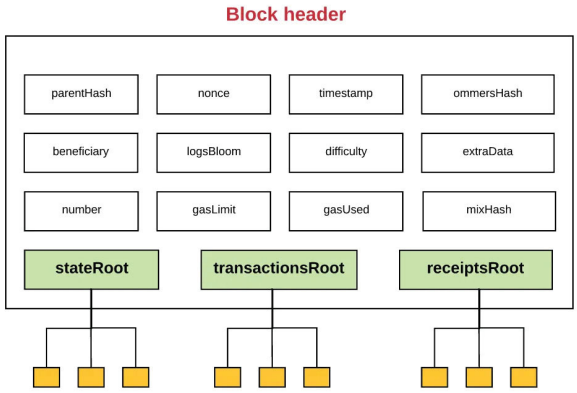

Fig. 1. Structure of Ethereum Block Header [5]

C. EVM

The Ethereum virtual machine (EVM) is deterministic, which means that it produces the same output every time it runs the same program. However, it can not determine if a program will ever end or continue indefinitely, leading to issues like denial-of-service attacks [6]. The EVM runs smart contracts, which are like programs on the EVM. Fees are charged to prevent issues caused by the deterministic machine, which will be explained later [7].

D. Smart contracts

Smart contracts are programs executed on the EVM with associated data and states. They can modify the state of Ethereum accounts, and all state changes made by calling such contracts are registered to the state of the Ethereum network. The contracts are isolated from the system in which they are executed and are mainly programmed in the language Solidity, which is widely used and popular due to its object-oriented principles [8].

E. Solidity

Solidity is the language used to develop smart contracts on the Ethereum blockchain. It has similarities with JavaScript and coding language C. It is an object-oriented, case-sensitive, static programming language. Compilation results in byte code, and it has automatic control of syntax errors. The language was developed by Gavin Wood, who is the co-founder of the Ethereum network [9]–[11].

F. Fees

Fees were implemented in the EVM to solve the issue of determinism, and these fees are calculated using gas, which is an estimate of the gas required for a successful transaction based on EVM instructions. The gas limit is initially set at 21,000, and each gas costs 0.000000001 ETH. However, when executing complex transactions that require multiple changes and state-saving operations, such as creating contracts, the gas fee increases to compensate for the time and computational power needed to solve them. The gas helps prevent denial of service attacks by limiting the attacker's ability to create infinite loops, and the cost of each transaction step is proportional to its complexity [11].

G. State channels and off-chain.

Off-chain solutions involve moving some transaction processing off the blockchain and onto a separate network or platform because of scalability and reduced fees. State channels are temporary channels created between parties for a specific purpose, where multiple off-chain transactions can be conducted without involving the blockchain for each one. The final state of the transactions is recorded on the blockchain once the channel is closed.

The Ethereum network charges fees for transactions, which can be problematic for games because the dollar value of the fees can be very high. For example, a $5 bet on roulette may not make sense if the transaction fee is $20. The state channels were created to solve this issue and operate off-chain, meaning they are not directly connected to the Ethereum network and are considered to be outside of the blockchain. Off-chain solutions allow faster and cheaper computation of results, increased privacy and scalability, and many more transactions at a lower cost per transaction. However, the main challenge with state channels is establishing trust in the off-chain network. One or more nodes verify that everything is fair and square to ensure safety. The zero-knowledge mode ensures the result is correct without revealing any other information. The main point behind state channels is to have ideally two on-chain transactions, one to open the channel and the other to close it, with numerous transactions occurring off-chain. This approach significantly reduces the number of on-chain transactions and associated fees [12], [13].

III. Related work

In this section, we would like to introduce related DeFi gaming platforms built on state channels in more detail. Finally, we will compare the benefits and drawbacks of each approach.

A. Ethex.bet

Ethex is an Ethereum smart contract lottery with simple rules. You have to guess 1 to 6 characters of the block's hash where your bet takes place. It operates fully on-chain, as it bets on the on-chain results. The minimum bet on their site is 0.01 ether per character. If you want to bet and try to guess just one character, the minimum bet stays at 0.01 ether. However, if you want to guess all six characters, the minimum bet increases to 0.06 ether. Validation is done in the following way: they add to the stack the number of the block, the bet amount, the ID of the transaction, and the bet-string array containing the characters being guessed. They even implemented a demo version that does not require a wallet or web3 connection. The only downside is that the EVM has access to only the last 256 block hashes. If your bet is not confirmed within 256 blocks, it will fail [14].

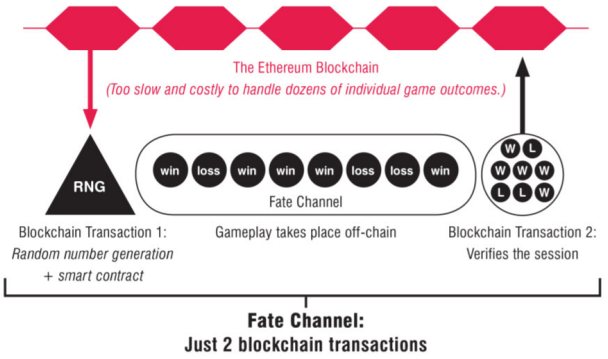

Fig. 2. Funfair off chain state channel concept [15]

B. Funfair

Funfair is a large gaming platform focusing on casino games and is fully built and implemented on the Ethereum network.They implemented their own ERC20 token, FUN, for betting and cashing out within their platform. However, it has been found that the average blackjack bet is only around a dollar, which means they needed to find a way to eliminate Ethereum blockchain fees that are not even close to $1. Another problem is that they would have to wait for multiple blocks to finish one bet. It could take time because the new block is mined every 10 to 15 seconds. Funfair introduces an off-chain solution with a state channel called the Funfair Fate Channel to solve these issues. We can see their idea of the Fate Channel shown in Fig. 2, where only two transactions are needed for creating and closing the channel. Games are played off-chain, and initial and final states are stored on the blockchain [15].

C. Sunrise Casino DAO

Sunrise is a gambling platform focusing on table games like poker and blackjack. They faced two main challenges in their design how to create their token and how to assign value to their tokens. They wanted to create tokens that operate like shares and retain value even after their distribution. They also had to solve the issue of the pseudo-random function. Initially, they planned to run their project on the Ethereum blockchain but, due to high fees and unsolved problems, they decided to use a private blockchain that would interact with web3. They will use the synthetic United States dollar (sUSD) token, which will be exchanged for Tether (USDT), a stablecoin fixed to $1. The choice of the private blockchain was influenced by the problems they faced on the Ethereum network [16].

D. Stixex

Stixex is a gaming platform that allows playing only one-player mode while betting on the odds of a trading pair on the market. The first problem with this idea is that betting on the commodity rate, namely Ethereum in US dollars, must be drawn from the centralized solution offered by the API. The problem arises when, for some reason, this centralized service stops or sends incorrect data. The highlight of this platform is that they use off-chain transactions, which are free of charge. A state channel is created for this. The transparency of a smart contract ensures fairness between participants. The site communicates via web3 on the Ethereum blockchain and does not offer the possibility of registration. In the process of tokenization, they work with USDT Tether, which has a fixed value to the US dollar [17].

TABLE I. COMPARISON OF DEFI GAMING PLATFORMS

Name	Ethex.bet	Funfair	Sunrise	Stixex	Degens
Blockchain	ETH	ETH	ETH	ETH	ETH
Whitepaper	No	Yes	Yes	No	No
Tokens	ETH	ETH,FUN	ETH,SUNC	ETH	DAI
Provably fair	Yes	Yes	No	Yes	Yes
Offchain	Yes	Yes	No	Yes	Yes

E. Degens

A gaming platform designed primarily for player-to-player betting. The authors created their token, DEGENS, which is used for several functions within their platform. Players can use Ether or DAI tokens (stablecoin on the Ethereum) for their bets, which are fixed to the price of US dollars. The platform uses off-chain methods to evaluate matches, which helps to save on fees and increase privacy. In order for a bet to take place, two players must be paired up - one betting for a particular outcome and the other betting against it. Pending games are listed on the login page, waiting for a player to join and place the bet [18].

F. Summarization of platforms

The compared platforms have several weaknesses but also several strengths. The most suitable platform is Funfair, which makes very good use of several elements. We would like to highlight their off-chain channel, which stores intermediate results of the game. On the other hand, the Stixex platform has a similar implementation with API centralization problems and outside blockchain solutions. Despite the off-chain solution, the FUNFAIR platform offers the possibility of tokenization and a large number of games. Other platforms have several problems, out of which we can highlight the Degens platform's only partially transparent verification of results. It can be considered a significant centralization, and the Degens creators agree. They justify it by saying that if they did not evaluate the correct results, everyone would stop using them, and no one would play with them. In the case of the Sunrise casino, players are almost forced to use platform tokens without being able to play directly with Ethereum coins, where you would reduce your fees and even have more confidence in the value of the coin. We selected several features of DeFi gaming platforms and compared then in Table I.

IV. DESIGN

As mentioned above, some projects are trying to implement DeFi gaming platforms using state channels to reduce transaction fees in the public blockchain network. Thus, we

will attempt to compare our project with those solutions' specifications. In contrast to the aforementioned platforms and what we have learned, we want to design a game platform with several games within the blockchain. The aim is to design several implementation types and point out at their advantages and disadvantages. The result is an effort to implement two types of games, one that takes place directly on the blockchain without off-chain transactions and another that takes place off-chain using state channels. However, we want to stress the benefits of the second layer, which offers high scalability. It saves transaction fees when we deploy our decentralized application into a mainnet network. The game begins with players A and B, who access the application's web interface and choose the game. If they are connected, they can start playing. The game does not have a time limit, but it is recommended to set a timeout in case one player stops playing or faces circumstances that prevent them from continuing. The channel should only be opened after adding both players. Opening the channel with only one player can cause implementation issues. For example, if player A opens the channel but can not find their opponent, they will have to close it with a value of 0, which does not make sense. Each player should be able to quit the game and close the channel anytime. We have implemented two games to demonstrate our solution and use different approaches to validate the final state for each of them, which are described in more detail in one of the following subsections. Now, let us introduce the most important components of our design.

A. Connecting players

One of the challenges we faced was how to establish connections between players. There are various approaches to addressing this issue, and one of the simplest is to pair players randomly. However, in our case, we wanted to retain the ability to select an opponent. Therefore, we developed a list of all active players waiting for a game, where players can sign up to join and start playing. To differentiate players, we use public keys that are unique to each player, giving each player a distinct identification. We have also implemented multiple functions, such as deploying the contract to initiate the game, which is done by Player A, and allowing Player B to join the game. Additionally, we have introduced features such as claiming a timeout, surrendering, and more. In Fig. 3, we can see the web interface for connecting to the game. After players select the game, they are invited to connect using the Metamask wallet.

B. Smart contact

For playing the game, the smart contract needs to include essential information. Most games involve two players, so it is necessary to keep important data about these two players. Specifically, we need to store information about the credit of Player 1 and Player 2, the public addresses of their wallet to identify the player, and whose turn it is. In terms of the game itself, it is critical to keep the round number being played, with each opponent's move counted as one round and the number iteratively increased from zero. It is also necessary to

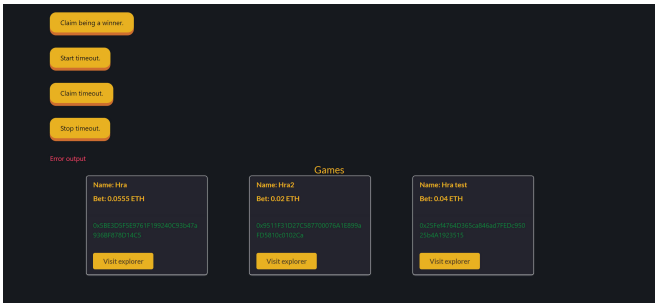

Fig. 3. Player Dashboard

identify the game being played, as there can be multiple games with varying rules, numbers of players, maximum rounds, and other factors. Consequently, we must verify the validity of the provided results when players attempt to close the channel. We save the timestamp, ideally when writing in the block or shortly before. Timestamp is also recorded if one or both players stop playing during the game. It is crucial to record and remember the bet value, for example, 0.0754814 ETH. Finally, the smart contract should include information about the game status. We define the following statuses:

- status number 0 is terminated normally
- status number 1 is terminated by the expiration of time
- status number 2 is terminated by the user
- status number 3 is an pending game

Fig. 4 describes the interaction between players and also shows the interface of our implemented smart contract. The following functions are defined there:

- **returnWinner** - determine the winner of the game after it finishes
- **getMessageHash** - return Keccak256 hash of received message from the opponent
- **verify** - declare the winner, verify the result, and also pay the reward to the winner
- **timeoutChallenge** - set the timer and challenge the player's opponent to play round
- **claimTimeout** - player, who invoked *timeoutChallenge*, calls it to pay off the reward after the timer is expired
- **cancelTimeout** - opponent calls it in response to *timeoutChallenge* before the timer is expired
- **recoverSigner** - validate sender of the received message
- **splitSignature** - divide a given board hash into 3 parts
- **getP1 and getP2** - return the address of the given player, if the return value is null, the game has not started yet
- **join** - joining player2 to the game, his bet value must be the same as the bet of player 1

C. State channel

Each game instance will be represented using a structure that will keep information such as who is their turn to play, the sequence number, and the state of the game, several constants. The game is closed by closing this channel, and the game statuses are written using the transfer function. The off-chain

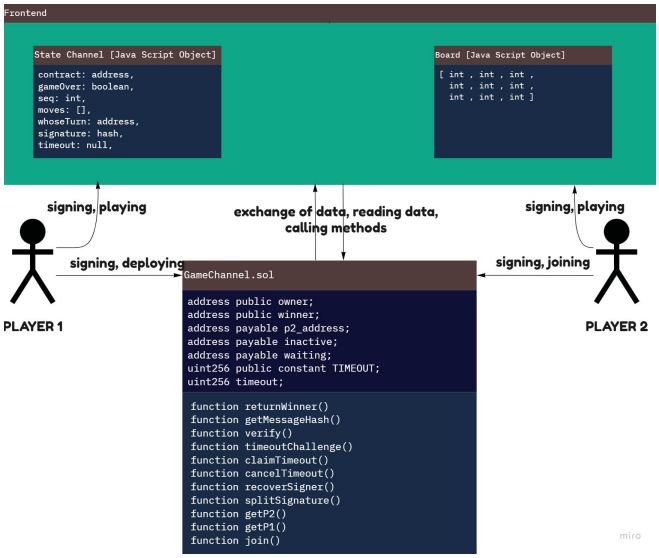

Fig. 4. Interface of Smart contract

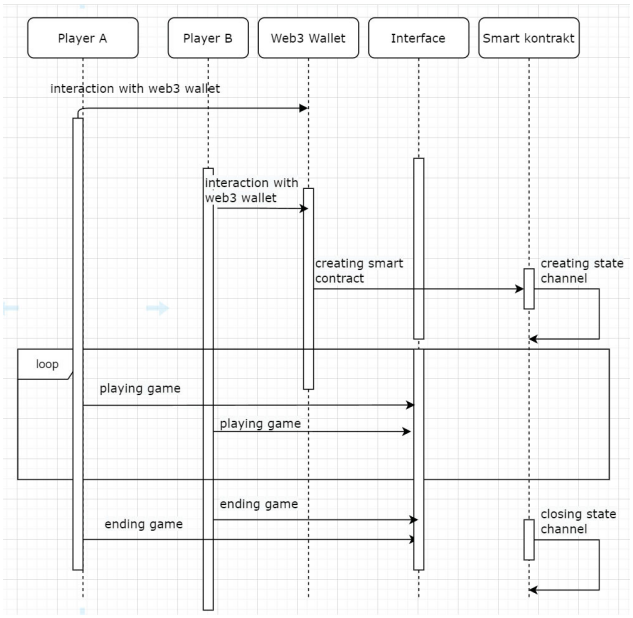

Fig. 5. Sequence diagram showing game without problems.

channel between players is created using the public service PubNub [19]. It allows exchanging of messages between players out of blockchain. The whole process of playing the game is shown in Fig. 5.

D. Verifying game states

Our implementation offers two methods to verify that a game session has been completed correctly and without cheating. The first method, demonstrated using the example of Tic-Tac-Toe, involves signing the game state by both players. Before the game starts, the smart contract saves the initial state of the game boards of both players, which must be encrypted to prevent the opponent from seeing them. Using an off-chain channel, players take turns making their moves until one player reaches the final state. If one player wins, the other has to sign their game board to declare their loss and send it to the winner, who also has to sign it. Once both players sign the final state of the game, known as the finishing move, the expected winner can call the function *claimWinner* and pass the signed game board to the smart contract to declare the winner and payout the bet.

The second method is based on implementing the Merkle tree for the game Blackjack. The game is usually played with 52 cards. Our algorithm will generate an array of all cards, sort them randomly, and make the root of the Merkle tree where all leaves will be cards. Players cannot change the order of the cards or cheat in any way, as they must provide proof that builds the root hash and compare it with the saved root hash. Failure to do so would be considered cheating.

Both methods allow players to invoke the function *timeoutChallenge* if their opponent appears inactive in the game. The opponent must acknowledge their presence in the game by calling *cancelTimeout* in response. If the opponent fails to respond, the timer will expire, and the requesting player can claim a reward by calling *claimTimeout*.

V. Evaluation

We defined several test scenarios to evaluate all the possible cases that can occur during the game, including a standard playing test, a cheating test, and even a timeout test. With these scenarios, we can observe what happens if one of the players decides not to play or can not continue playing.

Standard game scenario:

1) Player A joins the game.
2) Player B joins the game.
3) Both play at least a few rounds.
4) One of them loses.
5) The channel is closed, and the prize is paid to the winner.

Cheating game scenario:

1) Player A joins the game.
2) Player B joins the game.
3) Both play at least a few rounds.
4) One of the players tries to submit the end of the game with a fake signature.
5) If the system detects such behavior, it will punish the given player by giving everything to the opposite player, making him the winner.
6) The channel is closed, and the prize is paid to the winner.

Timeout scenario:

1) Player A joins the game.
2) Player B joins the game.
3) Both play at least a few rounds.
4) If one of the players stops playing, the opposite side triggers a timeout using the smart contract.

```
λ npx hardhat test
Compiled 1 Solidity file successfully

  GameChannel
    √ Deployed contract with 0.02 ETH value. User 2 joined successfully. (1439902 gas)
    √ Simulated game where player 1 should be a winner. (1449412 gas)
    √ Simulated game where player 2 should be a winner. (1474985 gas)
        Waiting 60 seconds...
    √ Simulation of timeout situation. (1561279 gas)

·-------------------------------------|---------------------------|-------------|-----------------------------·
|       Solc version: 0.8.4           ·  Optimizer enabled: true  ·  Runs: 200  ·  Block limit: 6718946 gas  |
·-------------------------------------|---------------------------|-------------|-----------------------------·
|  Methods                                                                                                    |
·---------------|---------------------|-------------|-------------|-------------|--------------|-------------·
|  Contract     ·  Method             ·  Min        ·  Max        ·  Avg        ·  # calls     ·  eur (avg)  |
·---------------|---------------------|-------------|-------------|-------------|--------------|-------------·
|  GameChannel  ·  claimTimeout       ·          -  ·          -  ·      52080  ·           1  ·          -  |
·---------------|---------------------|-------------|-------------|-------------|--------------|-------------·
|  GameChannel  ·  join               ·          -  ·          -  ·      43724  ·           5  ·          -  |
·---------------|---------------------|-------------|-------------|-------------|--------------|-------------·
|  GameChannel  ·  timeoutChallenge   ·          -  ·          -  ·     101995  ·           2  ·          -  |
·---------------|---------------------|-------------|-------------|-------------|--------------|-------------·
|  GameChannel  ·  verify             ·      67781  ·      69297  ·      68539  ·           4  ·          -  |
·---------------|---------------------|-------------|-------------|-------------|--------------|-------------·
|  Deployments                        ·                                         ·  % of limit  ·             |
·-------------------------------------|-------------|-------------|-------------|--------------|-------------·
|  GameChannel                        ·          -  ·          -  ·    1294183  ·      19.3 %  ·          -  |
·-------------------------------------|-------------|-------------|-------------|--------------|-------------·

  4 passing (1m)
```

Fig. 6. Gas Optimization with Hardhat-Gas-Reporter

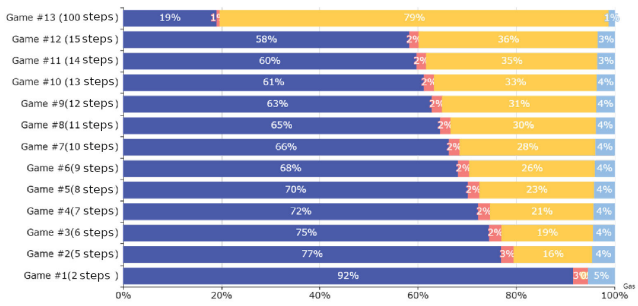

Fig. 7. Reduce of cost using the state channel

5) If the system detects such behavior, it will pay out the current balances of both players to the player who requested the check.
6) The channel is closed, and the prize is paid to the winner.

We performed multiple security tests on Linux and ran gas optimizations to reduce the cost of transactions. Fig. 6 shows the average cost of each called function in our smart contract in the test scenario when we played the Tic-Tac-Toe game. The parameter *run* is worth mentioning, which means how many times you expect functions to be called in that smart contract. If it is set to a lower value, it optimizes the byte code of the smart contract to be cheaper for the deployment and later function executions to be more expensive. On the other hand, deploying the smart contract is more expensive if it is set to a higher value, but the later function executions will be cheaper. On the bottom line of Fig. 7, we can see that deployment will cost 92 percent of the total cost for the smart contract, while the steps will cost next to nothing. However, looking at the top line, we can see that the yellow bar represents 79 percent of the total cost, which is the cost of moves. This test was performed for the game with 100 moves submitted to the blockchain. Using a state channel can significantly reduce fees based on our test results. The color labels represent the following information:

- blue - deploying of the smart contract
- light red - creating of the state channel
- yellow - off-chain transactions on the state channel
- light blue - closing of the state channel and verifying of game results

VI. Conclusion

In this paper, we have introduced the concept of a DeFi gaming platform using the state channel. It means that the initial and final state of the game is saved in the blockchain. Otherwise, the game is played by players off-chain, and the intermediate game results are not saved to the blockchain. This approach significantly reduces transaction fees. However, we have also implemented two different approaches to validate the game results for the game Tic-Tac-Toe and Blackjack.

An important aspect was validating the solution. We defined test scenarios for various game situations, including standard gameplay, cheating, and timeouts. Next, we focused on reducing transaction fees and confirmed that state channels help lower costs for players in the DeFi games.

We could not compare our design with existing solutions because most of them no longer exist or do not have code in public.

We aim to improve and expand the current implementation in the future. One of our key goals is to transform it into a plug-and-play gaming platform, but this will require significant effort because we can not validate results with the same approach for every game.

Acknowledgment

This publication has been written thanks to the support of the Operational Programme Integrated Infrastructure for the project: Research in the SANET network and possibilities of its further use and development (ITMS code: 313011W988), co-funded by the European Regional Development Fund (ERDF).

References

[1] G. Wood *et al.*, "Ethereum: A secure decentralised generalised transaction ledger," *Ethereum project yellow paper*, vol. 151, no. 2014, pp. 1–32, 2014. [Online]. Available: https://files.gitter.im/ethereum/yellowpaper/VIyt/Paper.pdf

[2] Proof-of-stake (pos). Accessed: 2023-04-26. [Online]. Available: https://ethereum.org/en/developers/docs/consensus-mechanisms/pos/

[3] P. Sajana, M. Sindhu, and M. Sethumadhavan, "On blockchain applications: hyperledger fabric and ethereum," *International Journal of Pure and Applied Mathematics*, vol. 118, no. 18, pp. 2965–2970, 2018.

[4] S. Bragagnolo, H. Rocha, M. Denker, and S. Ducasse, "Ethereum query language," in *Proceedings of the 1st International Workshop on Emerging Trends in Software Engineering for Blockchain*. Gothenburg Sweden: ACM, May 2018, pp. 1–8.

[5] A. Rishi. The ultimate guide to ethereum blocks. Accessed: 2023-04-26. [Online]. Available: https://blog.cryptostars.is/the-ultimate-guide-to-ethereum-blocks-98da8e2c1697

[6] S. Tikhomirov, "Ethereum: state of knowledge and research perspectives," in *Foundations and Practice of Security: 10th International Symposium, FPS 2017, Nancy, France, October 23-25, 2017, Revised Selected Papers 10*. Springer, 2018, pp. 206–221.

[7] Y. Hirai, "Defining the Ethereum Virtual Machine for Interactive Theorem Provers," in *Financial Cryptography and Data Security*, M. Brenner, K. Rohloff, J. Bonneau, A. Miller, P. Y. Ryan, V. Teague, A. Bracciali, M. Sala, F. Pintore, and M. Jakobsson, Eds. Cham: Springer International Publishing, 2017, vol. 10323, pp. 520–535, series Title: Lecture Notes in Computer Science.

[8] J. Toroman, "Application of ethereum smart contracts in purpose of generating new cryptocurrencies," *Global Journal of Computer Science and Technology*, 2018.

[9] S. Bragagnolo, H. Rocha, M. Denker, and S. Ducasse, "Smartinspect: solidity smart contract inspector," in *2018 International workshop on blockchain oriented software engineering (IWBOSE)*. IEEE, 2018, pp. 9–18.

[10] R. Modi, *Solidity Programming Essentials: A beginner's guide to build smart contracts for Ethereum and blockchain*. Packt Publishing Ltd, 2018.

[11] C. Signer, "Gas Cost Analysis for Ethereum Smart Contracts," p. 39 p., 2018, artwork Size: 39 p. Medium: application/pdf Publisher: ETH Zurich.

[12] J. Eberhardt and S. Tai, "ZoKrates - Scalable Privacy-Preserving Off-Chain Computations," 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)*. Halifax, NS, Canada: IEEE, Jul. 2018, pp. 1084–1091.

[13] J. Stark, "Making Sense of Ethereum's Layer 2 Scaling Solutions: State Channels, Plasma, and Truebit," p. 17.

[14] Ethex faq. Accessed: 2023-04-26. [Online]. Available: https://ethex.bet/faq

[15] Funfair. Accessed: 2023-04-26. [Online]. Available: https://funfair.ventures/

[16] Sunrise gaming by dao. Accessed: 2023-04-26. [Online]. Available: https://sunrisegaming-dao.com/pdf/Release_SUNRISECASINO_by_DAO_WHITEPAPER_v1.pdf

[17] Stixex. Accessed: 2023-04-26. [Online]. Available: https://stixex.io/#/

[18] Degens protocol documentation. Accessed: 2023-04-26. [Online]. Available: https://degensprotocol.github.io/degens-contract/protocol.html

[19] Pubnub. Accessed: 2023-04-26. [Online]. Available: https://www.pubnub.com/