Twenty-One: a state channel game

Roy Shadmon
Master's Project Report
rshadmon@ucsc.edu
University of California, Santa Cruz

March 26, 2019

Abstract

This paper is a Master's project report that discusses the implemented project, provides a security analysis, and reviews the author's accomplishments. The project was completed by the author for the Master's project requirement of his M.S. Computer Science degree at UC Santa Cruz. The resultant code is public on GitHub, and the link to the project repository is provided in the first footnote[1].

1 Introduction

The blockchain is a fairly new technology that has the means to change how two untrusting parties can verifiably play games, gamble, and send/receive money without the need of a centralized trusted third-party to arbitrate and enforce the process and the rules of the respective event. Unfortunately, using the blockchain as the trusted third-party for every single event is unfeasible, because every transaction costs money to make and takes some time to process. At the time of writing this paper, the average Ethereum transaction costs approximately $0.08 USD [1], however, the transaction cost on the Ethereum blockchain is based on the number of computation cycles a transaction takes to complete. For a game-based Smart Contract (such as this project), the transaction cost would be much higher than the average transaction fee[2][3]. In addition, if two users were to play a game through a Smart Contract fully on-chain, they would each have to wait approximately 3-5 minutes for each transaction to be included in a block and have there be enough appended blocks to highly guarantee that the block containing the respective transaction will remain on the longest chain[4].

This Master's project presents efficient and cost-effective techniques to make interacting with a Smart Contract more affordable–while also not having to wait the long confirmation time. For example, a party gambling 0.1 Ether[5] would want to minimize the fees they pay to maximize their net profit. By taking advantage of state channels (discussed in Section 2.3), we are able to minimize the total number of on-chain transactions to at most 2 transactions for a winning party and 1 transaction for a losing party. Moreover, state channels also allow us to not need to wait the confirmation time of on-chain transactions.

The presented distributed application (dApp)[6] is a relatively simple game, however, implementing the game to be played off-chain proved to be the majority of the work and deemed to be a significant challenge because of the project requirements:

- Any transaction to the dApp should always, if possible, be made off-chain[7], however, the dApp should recognize when to make the user commit an on-chain transaction.

[1]Project Repository: https://github.com/royshadmon/21-A-dApp-Off-Chain-Game
[2]For example, a transaction that transfers money to another address is cheaper than iterating through a large list on a Smart Contract.
[3]Section 1 discusses the cost of playing the game Twenty-One on chain.
[4]The longest chain is the valid ledger.
[5]Ether is the currency that lies on top of the Ethereum blockchain that holds real world value.
[6]This project is a web-based application that allows the user to interact with the Smart Contract.
[7]In certain situations an on-chain transaction is required, which will be more clear in Section 3.1

- A users' game move should be sent to their opponent in real-time.
- Only during the respective users' turn should that user be prompted the game buttons to make the next move.[8]
- A party should always know the most recent state of the game.
- A player must only accept their opponent's move if their move is verifiably valid.
- There should be a friendly and simple user-interface (UI).

2 Project Definitions

Before diving deep into the dApp game, it is important to define what the blockchain is, what are Smart Contracts, and what are state channels. Understanding these 3 terms are crucial to understanding the project and of its importantance.

2.1 The Blockchain

The blockchain is a public decentralized append-only ledger, also known as a state transition system, that keeps track of the ownership status of the underlying currency[9] [2]. The blockchain's security is based on digital signatures, and corrupting this security is a NP-hard task because integer factorization cannot be done in polynomial time. Moreover, many nodes verify the same transactions of a block and the majority of the nodes must agree on the validity of all transactions for each block to be added to the longest chain. Once a new block is added to the longest chain, the blockchain state is considered updated.

The Ethereum blockchain also supports a Turing-complete programming language called Solidity that is used to program Smart Contracts. In addition, the Ethereum blockchain is able to support, store, and modify the most recent state of a Smart Contract.

2.2 Smart Contracts

Smart Contracts are self-executing scripts that allow for dynamic, real-time execution of code triggered by transactions to the network [2]. The code is public and anyone can verify and read the code prior to engaging with the contract. The network provides the guarantee that the contract will be explicitly followed as it is written.

More specifically, Smart Contracts published on the blockchain have their own address. A function in the contract can either be triggered by addressing a transaction to the function or through data-driven events. Each time a contract is triggered, the transaction triggering the the contract is recorded on the blockchain. Contracts can contain an access control list (enforced by crpytographic signatures) that restricts who or when a person can make a transaction to trigger a function. For example, a contract can express the following type of logic: "If it is not player 1's turn, and player 1 is a player in the game, then player 1 can trigger a timeout on player 2".

2.3 State Channels

A state channel is a broad term used to discuss what could theoretically occur on the blockchain. Since every event recorded on the blockchain is digitally signed, state channels are digitally signed events that could but aren't recorded on the blockchain. What makes state channels special and promising is that two untrusting users could send off-chain signed messages to each other, and if there is ever a dispute, each user could submit their most recent state of events to the blockchain. Since each message is digitally signed (and could have only been signed by the respective sender), the blockchain would verify each message and be the third-party who determines what the correct and most recent state is.

[8]This prevents a user from making an invalid on-chain transaction, which saves them from paying the transaction fee of an invalid transaction.
[9]On the Ethereum blockchain, the underlying currency is Ether

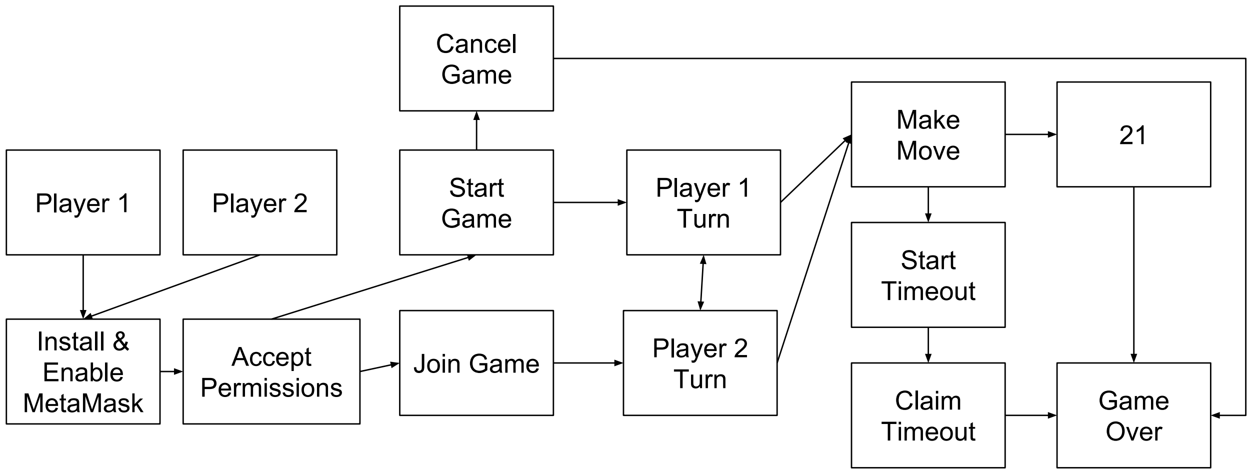

Figure 1: The flow chart of the 21-dApp game of two players: player 1 and player 2.

State channels have proved to be an important break through to alleviate the need to pay transaction fees per transaction, and they also prevent the need to wait the confirmation time to be ensured the transaction succeeds. As long as a user can verify the message in accordance to the rules of the Smart Contract, then that user can be assured that the signed off-chain message will be approved by the network if it were to propagate the transaction to the network.

3 21: The Game

The game Twenty-One is a turn-based game involving two players who take turns incrementing a counter (starting at 0) by either 1, 2, or 3 where the first player to increment the counter to 21 wins. The dApp is hosted on the internet (as a web application) and the official backend is the Ethereum blockchain, where users interact through the dApp to play the game. In addition, each new game is a new Smart Contract (containing the same code), which is done for simplicity reasons and to prevent the Smart Contract meta-data from becoming too large (which would inflate the transaction fees).

3.1 Game Flow

As shown in Figure 1, each player must first install and enable the MetaMask browser plugin[10]. Once this step is done, the user is then prompted by the dApp for permission (read-only) to view the addresses and respective balances stored in the user's MetaMask wallet. Once accepted, the user is then able to start a game (on-chain), where they will receive the Smart Contract address of the game. That user (player 1) would then need to send the contract address to their desired opponent (player 2), who would join the game (on-chain)[11]. Prior to player 2 joining, player 1 is able to cancel the game (on-chain) and withdraw their deposit from the Smart Contract, however, they cannot cancel the game once player 2 joins. Moreover, after player 2 joins, player 1 (who starts) and player 2 alternate making moves (off-chain), which in game theory is the rational way to play. If either player is taking too long to make a move, then the opposing player can start a five-minute timeout (on-chain) against the opposing player[12]. If the opposing player doesn't respond with an on-chain move during the timeout period, then the waiting player is able to claim the timeout (on-chain) and be declared the winner of the game, where they also receive the balance of the contract. If the players are responding quickly, then they keep exchanging signed

[10]MetaMask is a browser wallet.
[11]If player 2 started a game, there would simply be a new contract address where they are player 1.
[12]Starting a timeout costs a transaction fee.

messages (off-chain). Whichever player increments the counter to 21 wins, and commits an on-chain transaction to the blockchain where they are officially declared the winner and are transferred the contract's balance.

3.2 Signed Messages

Signed messages (also known as state channels) only work when the users who interact with the contract act rationally, honestly, and timely. If one player ever deviates from the protocol of the game, then the game must be played on-chain. It is also important to note that the Smart Contract will only update to a state that is signed by the opponent of the player submitting a state to the Smart Contract.[13] We will discuss the threat model and the different types of players to our game in Section 4, however, for now assume that all players are acting rationally and follow the game's protocol.

After player 1 starts the game, and player 2 joins the game, player 1 sends player 2 a message containing verifiable proof of their move. The message contents are as such: the current contract address, the sequence number, and the new counter value. Player 1, in addition, hashes and digitally signs the message, which is the digital signature (verifiable proof) player 1 actually made the respective move[14]. Upon receiving the message and the digital signature, player 2 must verify that the signature is valid, and if so update its most-recent off-chain state. If the move is invalid, player 2 rejects the move and waits for player 1 to send a new move. This process repeats until a player is able to update the counter to 21.

4 Threat Model

We classify the majority of our users to be honest and rational (following the protocol), because being dishonest and irrational would be financially costly and is not the rational way to play from a game theory perspective.

There are a few viable attacks to this dApp that we have taken significant precaution and protect our users in our implementation of this game. We will discuss three possible attacks and how we prevent these attacks from occurring.

4.1 Attack 1: Not Interacting with the dApp

The first attack we discuss is an attack where a player sends an on-chain timeout transaction independently and not through the dApp. The problem with this attack is that if a player starts a timeout against their opponent through the dApp, then their opponent is immediately notified that a timeout was made and their next move on the dApp will be made on chain. For example, if the timeout was started by player 1 on player 2 without interacting with the blockchain, then player 2 could be oblivious to a timeout being placed on them. Each player could continue sending off-chain messages and once the timeout period ends, player 1 claims the timeout and automatically receives the contract balance. The problem here is that player 2 wasn't acting maliciously (since they were sending signed messages) and would have committed an on-chain move had they known that a timeout was placed on them.

We prevent this type of attack by implementing an automated, asynchronous thread that queries the blockchain every 10 seconds for on-chain events of the respective contract address, and we notify the player facing the timeout of the need for a timely move. We also make sure that dApp would recognize when a player needs to make an on-chain move and require the player to commit an on-chain move. This is an important security patch, because it will ensure that an honest player who makes timely moves will not be a victim of this attack.

4.2 Attack 2: Replay Attacks

A replay attack for this game is when an adversary tries to reuse their opponent's signature from a previous game as proof of state for a different game.

[13] This is done to prevent a player from randomly signing and submitting an unacknowledged state of the game.

[14] Signing messages involves the user to sign the message using their private key. The blockchain's underlying security is that a user's private key remains secure, and that any signature that can be associated to an address was made by the owner of the respective address.

We prevent this type of attack by including the current contract address in the message hash so that the opposing player would reject the signed message, and more importantly, the on-chain Smart Contract would deem the signed message as an invalid state.

4.3 Attack 3: Refreshing Browser

Since the players have a secondary known communication platform[15], a nice player (player 1) prior to starting a timeout could ask their opponent (player 2), over their respective messaging platform, when they plan on making a move to avoid needing to pay for an on-chain timeout transaction. Player 2 could claim that they did make a move and advise player 1 to refresh their browser. If player 1 doesn't save their opponent's most recent signed state[16] and the most recent state of the game, then player 1 loses all evidence of the game state.

We prevent this type of attack[17] by saving in the player's browser memory their opponent's most recent move (signature, sequence number, and counter value). Therefore, upon refreshing and rejoining the game, the dApp will query the on-chain blockchain state and the browser memory of the most recent state of the respective player. The player's next move will be made from the most recent state resulting from the two queries.

5 Transaction Fee of dApp Functions

This section discusses the cost of on-chain transactions, where at the time of writing, the approximate price of 1 Ether in USD is $134.84 [3]. Figure 1 illustrates the cost of each function.

Function Name	Description	Price in Ether	Price in USD
Start Game	Starts the game	0.006	$0.81
Cancel Game	Cancels the game	0.001513	$0.21
Join Game	Joins the game	0.001918	$0.26
Start Timeout	Starts a timeout on opponent	0.000994	$0.14
Claim Timeout	Claim timeout on opponent	0.001537	$0.21
Move From a State	Move from an off-chain state	0.04	$5.40
Move On-Chain	Move from an on-chain state	0.00114	$0.16

Table 1: The price of different on-chain transactions for this dApp.

The two notable transactions are "Move From a State" and "Move On-Chain". The difference between the two functions is that moving from a state requires the miners[18] to verify the opponent's most recent off-chain move and signature, and then they must process player 1's next move on-chain. Moving on-chain simply requires the miners to verify that the move is valid from the previous on-chain state. Therefore, it is apparent that it takes significantly more computation cycles to verify an off-chain message and signature than just updating the state of a contract.

Some may wonder why not just make every move on-chain since it would be cheaper than having to make a single move from a state transaction? The answer to this is fairly simple: game theory says that the players are incentivized to follow the protocol, because deviating from the protocol is unnecessarily expensive. Since money is finite, it does not make sense for any player to act irrationally because there's no benefit to do so. Moreover, even playing a game using on-chain transactions would eventually overcome a player's available balance. Imagine playing a game where the total number of on-chain moves could be infinite (like in chess). Lastly, playing on-chain is slow and would require each turn in a game to have long wait-times.

[15]Since the player who starts the game must send the contract address to their perspective opponent.
[16]The digital signature, sequence number, and counter value.
[17]We acknowledge it's a naive attack, but it is extremely easy to make.
[18]The machines verifying all transactions and doing the proof-of-work.

6 Technologies Used

This project uses a few tools: Ganache, Truffle Suite, node package manager (NPM), Web3, MetaMask, and PubNub. Ganache is a personal blockchain mainly used for Ethereum dApp development. It makes Smart Contract deployment easy and efficient, and it allows developers to easily test their dApp. The Truffle Suite is the popular development environment for Ethereum dApp developers because it allows for easy and simple contract deployment and for its powerful interactive console. NPM works with Node.js, which is the popular JavaScript programming language used to develop dApps. It also works easily with the Truffle Suite. Web3 is the Ethereum JavaScript API that allows us to query the Ethereum blockchain or initiate transactions from a dApp. MetaMask is a browser wallet that allows a user to connect to a dApp. What makes it particularly useful is that it allows users to run Ethereum dApps in their browser without needing to run a full Ethereum node. Lastly, PubNub is a global messaging tool that allows us to send state messages to the respective user in real-time.

7 Future Work

Future work on this project includes developing additional games to publish to the community. Some projects of interest include: developing games that involve more than 2 players and games that can be played against an automated computer (or Casino). Moreover, there are lots of work and research being done on how to use state channels in other systems and decentralized protocols. It is becoming mainstream to implement blockchain technology in current technologies. Since many applications require real-time events, there are a lot of research opportunities to see how to best implement the blockchain and state channels in existing applications.

8 Conclusion

This Master's project work presents a dApp game that exhibits the importance of state channels and how they can minimize transaction fees and confirmation block times of untrusting interacting users–while maintaining an unknown, trusted, decentralized third-party to resolve any future disputes. In addition, this project shows that the blockchain is only needed in the case of a dishonest or irrational player.

State channels are of up-most importance in making use of the blockchain in any real-world application, which is why it is important to study and to understand thoroughly. This project allowed us to gain an extensive understanding on how to develop Smart Contracts that can be defined as off-chain states. Moreover, this project taught us how to develop an application that: uses state channels, can pragmatically verify states, and can pragmatically commit on-chain transactions through a dApp. Overall, understanding these technologies is a major stepping-stone into understanding the blockchain at an expertise level. In addition, we learned how to effectively use the blockchain, and how to design securely verifiable applications. This project is the start of our career into doing advanced research in blockchain and other related topics.

References

[1] BitInfoCharts, "Ethereum avg. transaction fee historical chart." https://bitinfocharts.com/comparison/ethereum-transactionfees.html.

[2] V. Buterin, "Ethereum: A Next-Generation Smart Contract and Decentralized Application Platform," *https://github.com/ethereum/wiki/wiki/White-Paper*, 2015.

[3] coinmarketcap.com, "Cryptocurrency market capital." https://coinmarketcap.com/.

[4] A. Miller, I. Bentov, R. Kumaresan, and P. McCorry, "Sprites: Payment channels that go faster than lightning," *CoRR abs/1702.05812*, 2017.