TRUSTED REAL-WORLD EVENT OUTCOMES ON A BLOCKCHAIN: STAKED VOTING AND ITS APPLICATION TO DISTRIBUTED GAMING

A Thesis
submitted to the Faculty of the
Graduate School of Arts and Sciences
of Georgetown University
in partial fulfillment of the requirements for the
degree of
Master of Science
in Computer Science

By

Tucker Spaight Moore, M.S.

Washington, D.C.
April 20, 2021
TRUSTED REAL-WORLD EVENT OUTCOMES ON A BLOCKCHAIN:
STAKED VOTING AND ITS APPLICATION TO DISTRIBUTED GAMING

Tucker Spaight Moore, M.S.

Thesis Advisor: Eric W. Burger, Ph.D.

ABSTRACT

In 2008, Bitcoin was introduced to the public and became the first widely accepted fully
decentralized cryptocurrency. Since the inception of Bitcoin, decentralized blockchain
technology has been utilized across many industries with incredible impact. However, as
effective as blockchain technology has proven to be, one challenge that exists is the ability to
bring real-world data into a decentralized network. Previous approaches rely on “trusted” feeds,
self-reporting with challenge mechanisms, and various, mostly centralized, oracles. All
approaches, though, have their own deficiencies. What is needed is a system and method of
authenticating real-world event outcomes across a decentralized network in a robust and low-cost
way.

One application for such a method is gaming. Casinos offer many benefits to a gambler.
They allow players to bet on games and events, determine event outcomes, and provide capital to
manage payouts. However, the players also face large fees, often masked as unfavorable odds, in
order to participate in this centralized process. It is in these high fees and unfavorable odds that
there exists an opportunity and need for a decentralized platform with the ability to provide
players with a more favorable means to engage in gambling.

As noted previously, a difficulty in creating a decentralized gambling platform is
enabling the network to reliably agree on the outcome of a real-world event, such as the winner
of the 2018 World Cup Final. Because a blockchain is a “closed-world” system, users can only access data that is created in that world. We seek to solve this problem by implementing a novel method of staked voting. Utilizing “actions” representing pools, bets, votes, and transactions, users can create pools representing real-world events, place bets on pools, cast votes on pool outcomes, and transact with other users. In order to ensure trusted outcome determination, we require winning bettors to vote on the outcome of another pool in order to redeem their winnings. If they vote with the consensus, their winnings will be released. If, however, they vote against the consensus, their winnings will be forfeited. This method of staked voting creates a fully distributed, self-regulated, pari-mutuel gambling platform.
## TABLE OF CONTENTS

Chapter 1: Introduction ........................................................................................................... 1

1.1 General Background ......................................................................................................... 1

1.2 Motivation .......................................................................................................................... 1

1.3 Contribution ...................................................................................................................... 2

Chapter 2: Background ......................................................................................................... 4

2.1 Cryptocurrency Background ............................................................................................. 4

2.2 Oracles ................................................................................................................................ 8

2.3 Deficiencies of Prior-Art ...................................................................................................... 16

2.4 Gambling Background ........................................................................................................ 17

Chapter 3: The Fortuna Approach ...................................................................................... 20

3.1 Introduction ....................................................................................................................... 20

3.2 Assumptions ....................................................................................................................... 21

3.3 Process Description ........................................................................................................... 21

3.4 Staked Voting ..................................................................................................................... 26

3.5 Voting Partitions ............................................................................................................... 28

3.6 Quorum .............................................................................................................................. 31

3.7 Payout ................................................................................................................................ 32

Chapter 4: Implementation .................................................................................................... 34
4.1 Balances .......................................................... 34
4.2 Fees .......................................................... 35
4.3 Tyche .......................................................... 35
4.4 Bootstrapping and the Genesis Block ........................................ 36
4.5 Action .......................................................... 37
4.6 Validation ...................................................... 42

Chapter 5: Results and Discussion ................................................. 47
5.1 Results ......................................................... 47
5.2 Discussion ..................................................... 48

Chapter 6: Conclusion and Future Work ........................................... 53
6.1 Summary and Conclusion .................................................... 53
6.2 Future Work ..................................................... 55

REFERENCES .......................................................... 57
LIST OF FIGURES

Figure 1. Advantages and Disadvantages of Existing Decentralized Oracles ................................. 10

Figure 2. Fortuna Process Description .............................................................................................. 25

Figure 3. Standard Staked Voting vs Fortuna Staked Voting .......................................................... 28

Figure 4. Fortuna Voting Partition Example ...................................................................................... 31

Figure 5. Winning Bettor Payouts .................................................................................................. 33

Figure 6. Class Diagram of Fortuna’s Action Protocol ................................................................. 37
LIST OF TABLES

Table 1. Class Parameters for an Action................................................. 38
Table 2. Class Parameters for an Action Payload .................................. 39
Table 3. Action Types and Corresponding Values ................................... 39
Table 4. Class Parameters for a Pool ....................................................... 40
Table 5. Class Parameters for a Bet ......................................................... 41
Table 6. Class Parameters for a Vote ....................................................... 41
Table 7. Class Parameters for a Transaction ............................................ 42
CHAPTER 1: INTRODUCTION

1.1 General Background

In 2008, Bitcoin was introduced to the public and became the first widely accepted fully decentralized cryptocurrency. It was the first cryptocurrency to combine blockchain technology with the concept of proof of work [1] to address the problem of double spending [2]. Since then, thousands of other cryptocurrencies have been created with almost one thousand trading daily [3]. However, as impactful as blockchain technology has been, a challenge that still exists is the ability to bring real-world data onto the blockchain in a reliable and trusted way.

Most blockchains are entirely self-contained, with no mechanism to validate data from the real world. Users can only access data that is reliably created on the blockchain [4] and the “truth” is limited to what the ledger can verify. However, there is a clear need for such a mechanism without a centralized point of failure and with the same security and veracity expected of a blockchain.

We hypothesize that, by implementing a novel staked voting algorithm and a unique pari-mutuel design, we can create a distributed protocol that satisfies the following: (i) no centralized point of failure; (ii) events with any number of outcomes are supported with the same level of trust; and (iii) resistance to expected attacks such as Sybil attacks, collusion, denial of service, and hash manipulation.

1.2 Motivation

Due to a growing distrust in the press and other centralized institutions [5], there is a growing need to bring real-world data into a blockchain in a reliable and robust way. However,
most decentralized blockchains do not provide users with the ability to validate the outcome of real-world events.

To address this problem, blockchains utilize oracles [6, 7]. There are many different types of oracles, but they all serve the same purpose – to transfer data between the real world and the closed-world system of the blockchain [8, 9]. However, a significant challenge in incorporating oracles on a blockchain, especially a decentralized blockchain, is maintaining trust across its users. There must be an incentive in place for each oracle to act honestly. While there are several oracle platforms [6, 10, 11, 12], the current state of the art has significant deficiencies: (i) there are major limitations on supported event types; (ii) reporting algorithms are susceptible to attacks; and (iii) the existing platforms exhibit varying degrees of centralization.

One application for such a network is gaming. Casinos offer many benefits to a gambler. They allow players to bet on games and events, determine event outcomes, and provide capital to manage payouts. However, the players also face large fees, often masked as unfavorable odds, in order to participate in this centralized gaming process. Nevertheless, there is still an estimated $150 billion wagered on sporting events each year [13]. Due to the size of this market and the need for a decentralized platform with the ability to provide players with a more favorable means to engage in gambling, gaming offered an intriguing use case for a distributed oracle network.

1.3 Contribution

The contribution of this thesis is Fortuna, a distributed oracle implementing a novel staked voting algorithm and a unique pari-mutuel design, as well as a prototype demonstrating and testing the protocol. Fortuna offers a decentralized platform on which real-world event
outcomes can be determined and provides users with a more affordable and favorable means to engage in gambling without the need for, or the costs of, a centralized authority.

Fortuna supports the use of actions, representing pools, bets, votes, and transactions. Users interact with the network by creating pools representing real-world events, placing bets on those pools, casting votes on pool outcomes, and transacting with other users. These actions allow users to interact with each other, and the protocol designed around them, including our novel method of staked voting and unique pari-mutuel design, creates a fully distributed, self-handicapping, gambling network that allows non-binary real-world event outcomes to be determined in a reliable and trusted way. Finally, Fortuna guarantees that non-binary event outcomes can be determined with the same level of trust as binary event outcomes while defending against attacks that existing oracle networks are subject to, including Sybil attacks, collusion, and denial of service.
CHAPTER 2: BACKGROUND

2.1 Cryptocurrency Background

A digital currency is a currency without a physical form and is only transferrable electronically [14]. Traditional digital currencies are centrally issued and regulated by banks or, in some cases, by privately owned companies. A cryptocurrency, however, is a type of digital currency that is typically decentralized, requiring no central regulating entity [2]. Decentralized cryptocurrencies use cryptography and a public ledger system to ensure honesty among users. Decentralization allows distributed security across its users and faster settlement of payment [15].

The first cryptocurrency, eCash, was proposed by David Chaum in 1983 [16]. eCash was a digital currency made available to customers of a bank as a way to anonymously withdraw money and later spend that money at participating merchants. eCash, however, only allowed coins to be transferred one time. Thus, once a merchant received payment, the coins were immediately deposited back in the bank [17].

In 2008, Bitcoin was introduced to the public and became the first widely accepted fully decentralized cryptocurrency. It was the first cryptocurrency to combine blockchain technology and “proof of work” in a way that provided a fully decentralized network the ability to timestamp transactions and prevent double spending [2]. “Double spending” occurs when a user places two transactions with two validating entities at the same time, referencing the same coin signature. Because each validating entity initially considers the transaction to be valid, both are accepted [18]. The blockchain, as a public ledger of all previous transactions, addresses the double spending issue. The blockchain’s ability to timestamp transactions by successively chaining
blocks of transactions together prevents two conflicting transactions from being published—
“chaining” requires one transaction to be processed before the other, thereby invalidating the
second transaction [19].

Since the inception of Bitcoin, thousands of other digital coins and tokens have been
created with a total market capitalization of approximately $340 billion and almost one thousand
trading daily [3]. In general, these decentralized blockchain based cryptocurrencies are structured
in similar ways and share standard practices and definitions. The following are definitions and
techniques generally incorporated into these cryptocurrencies.

2.1.1 Blockchain

A blockchain is a distributed ledger shared and synchronized over multiple entities or, in
the case of a cryptocurrency, over multiple users. This ledger documents and preserves the
history of the cryptocurrency by storing all previous transactions and is used to validate all future
transactions [20]. The blockchain stays consistent across the network by cryptographically
linking blocks of transactions. This linkage ensures the integrity of each block. If a block were
changed, or if a transaction within a block were changed, the hash of that block also would
change. Therefore, all subsequent blocks would need to change as well in order to remain valid.
This makes an attack on a previously mined block or transaction infeasible [21].

2.1.2 Proof of Work

“Proof of work” is what allows transactions to be mined into blocks and added to the
blockchain. Blocks are made up of validated transactions, a timestamp, the hash of the previous
block, and a nonce satisfying the proof of work requirement. A nonce is simply a value included with the block’s data that can be changed freely to produce different hashes. For a block to be considered valid, the proof of work requirement must be satisfied (i.e., the block’s data must produce a hash with a certain number of leading zeros) [19].

Proof of work was first proposed in 1997 as a method of throttling requests on otherwise unmetered systems [1]. Proof of work is used in cryptocurrencies in the same way. Requiring a non-negligible amount of work before allowing a miner to publish a block is a major deterrent in spamming the network with false blocks and transactions in the hopes that one gets through. The cost function typically used to show proof of work is designed to be “parameterizably expensive” [1] but requires only a single hash operation to validate. Once the miner determines this nonce, the block is broadcast to the network where it is added to the blockchain.

2.1.3 Nodes

A node is simply a user on the network. As in all decentralized networks, where there is no central server or entity [22], nodes on the network are responsible for connecting users and performing the tasks required for the network to run. Typically, there are three types of nodes on cryptocurrency networks: simple nodes (otherwise known as lite nodes), mining nodes, and full nodes.

A simple node is a user who initiates, or “publishes,” transactions to the network. To publish a transaction, a user will first sign the hash of a previous transaction along with the address (unique identifier assigned to a user) of the new owner and then broadcast that data to
the network through its neighboring full nodes. A full node is a node that validates new transactions and blocks as well as connects users across the network.

A mining node, or miner, is a user who constructs, or mines, blocks of validated transactions to help build the blockchain. The miner forms a block of new validated transactions and then performs proof of work. Once the miner has satisfied the proof of work requirement, they publish the block to the blockchain by broadcasting it to the network through its neighboring full nodes. For performing proof of work and building the blockchain, miners are rewarded with coins. These coins are not sent from another user but are “generated” by the block itself once it is published to the blockchain. This design incentivizes miners to act honestly. If a nefarious miner publishes a block deemed fraudulent by the network, future miners will not recognize that block and therefore its reward will be worthless.

Full nodes are responsible for validating new transactions and blocks, as well as disseminating those transactions and blocks across the network. Validating transactions and blocks maintains the integrity of the blockchain by ensuring users do not spend coins they do not have or publish blocks without first performing the required proof of work. The second responsibility of a full node is to disseminate transactions and blocks across the network. Most cryptocurrencies built on decentralized networks use a “gossip protocol” to disseminate data based on the way rumors are spread [23]. One full node sends data to its neighboring full nodes. Each of those full nodes then sends the data to their own neighboring full nodes, and so on. A neighboring full node is one with which another node can communicate freely. If a full node receives data that it has already received, the transmission stops. In the absence of a centralized server or entity, the network relies on each full node to disseminate messages. The way the
network is structured, as long as a node can communicate with at least one full node, messages will be successfully broadcast to the entire network.

2.2 Oracles

As impactful as blockchain technology has become, a challenge that still exists is the ability to bring real-world data onto the blockchain in a reliable and trusted way. Oracles help solve this problem. An oracle is a privileged entity with the power to transfer data between the real world and the blockchain.

There are many types of oracles, but most fall within three categories: software, hardware, and human [9]. A software oracle is simply software that pulls data from an online source (e.g., database, website, etc.) and injects that data into the blockchain. A hardware oracle is typically one that measures data itself before injecting it into the blockchain. This can be a temperature sensor, IoT device, etc., and is often used to execute a smart contract based on the measurement taken from the device. The last category is a human oracle. A human oracle is a user on a decentralized network with the ability to transfer data between the real world and a blockchain.

Additionally, there are subcategories: inbound/outbound, push/pull, and consensus based. Inbound/outbound describes which direction data is flowing. An inbound oracle is one that brings data into the blockchain from the outside world and an outbound oracle is one that transfers data from the blockchain to the outside world. Push/pull refers to the initiator of the transfer [8]. First, consider an inbound oracle. “Push” would refer to an entity outside the blockchain sending data to the blockchain itself, whereas “pull” would refer to an entity on the
blockchain requesting data from the outside world. And finally, a consensus-based oracle is one in which data determination is made based on multiple oracles as opposed to a single oracle.

A difficulty in incorporating oracles, especially human oracles, on a decentralized network is in maintaining trust across its users. There must be an incentive put in place for each oracle to act honestly. This is necessary to guarantee (with high probability) the consensus reached by a quorum of voters, or oracles, matches the “truth.” Without an incentive to vote honestly, there is no reason to believe a real-world event outcome can be correctly determined by a blockchain.

The following are four oracles we investigated further and Figure 1 details advantages and disadvantages of each. While each platform has their own unique design, they nearly all follow the same general process.

**Oracle Process**
1. Event proposed to the network
2. Bets placed (optional)
3. Event occurs in the real world
4. Oracles report outcome on blockchain
5. Rewards/payouts occur
2.2.1 ZenSports

ZenSports (Figure 1a) is a partially decentralized sports betting marketplace built as an application on the ICON blockchain [11].

ZenSports utilizes “matchmaking” to pair two users with opposing positions. The first step for a bettor is to select an event on which to bet. These are limited to binary outcome sporting events proposed by an internal ZenSports team. A binary outcome event is an event with only two potential outcomes. The bettor then selects their predicted outcome, the odds of the bet they wish to make, and the amount of the bet. This user is called the “maker.” A user who accepts a maker’s bet is called a “taker.” A taker can accept all, or any portion, of the maker’s bet.

Once the event has occurred, the maker has the responsibility to report the outcome. To deter the maker from misreporting event outcomes, the maker is also required to post a small escrow fee. If the taker disputes the reported outcome, the taker is required to post a small
escrow fee as well. At that point, ZenSports’ internal team resolves the dispute acting similar to a casino as the authority on event result determination [24].

While ZenSports provides a protocol to allow peer-to-peer betting, a large part of the protocol is operated by the ZenSports team. First, while the betting contracts themselves are created by users, they are limited to the events and bet types ZenSports chooses to support [25]. In addition, although there are plans to decentralize the process in the future, at this time, dispute resolution is handled unilaterally by ZenSports [26]. This centralization within the platform precludes users from proposing their own events representing less popular sporting events or non-sporting events such as the price of a stock on a given date or the winner of the Academy Award for Best Picture.

2.2.2 Wagerr

Wagerr (Figure 1b) is a semi-centralized gambling platform made up of two user types: “standard wallets” and Oracle Masternodes. Standard wallets interact with the network, validate transactions, and form blocks of transactions to build the blockchain and earn rewards. There are 2,000 Oracle Masternodes that are responsible for publishing events to the network as well as voting on event outcomes. Like ZenSports, Wagerr limits its events to certain binary outcome sporting events that are selected by consensus by the Oracle Masternodes [12].

After an Oracle Masternode publishes an event to the blockchain, users can begin placing bets. There are three ways to bet on an event: head-to-head, multi-user, and peerless direct chain betting.
In head-to-head and multi-user bets, Wagerr uses matchmaking to group users. First, a single user will place a bet on a previously created event, specifying the spread and the amount of the bet. Then, another user can “accept” that bet by placing a complementary bet. If a single user places a complementary bet that matches the amount proposed by the initial bettor, it becomes a head-to-head bet.

If a single user places a complementary bet for less than the amount proposed by the initial bettor, another user can “complete” the multi-user bet by placing their own complementary bet that covers all, or a portion, of the remaining bet amount. Once the pair or set of bets is determined to be complementary, a smart contract is created. After the event completes, the network of Oracle Masternodes votes on the outcome. If 1,500 of the 2,000 oracles vote on one outcome, the winners of the bet are paid out accordingly and the oracle who processed the payout receives half of the fees charged to the bettors.

In a peerless direct chain bet, a user can initiate a contract without an opposing side. In such a case, after the oracles have voted and the user wins the bet, the processing oracle will execute a smart contract paying out the user with newly minted coins. If the user loses, their coins will be destroyed. This methodology introduces a risk of the total coin supply growing or shrinking based on a series of either lucky or unlucky bets – minting a large number of new coins or destroying a large number of existing coins. However, over time, Wagerr believes this risk will balance out.

To encourage the value of WGR (Wagerr’s coin) to balance out over time, Wagerr destroys half of all fees paid out by the bettor: 2% on head-to-head bets, 4% on multi-user bets, and 6% on peerless direct chain bets. This behavior is based on the premise of “Value Coupling.”
Value Coupling is the idea that the value of WGR is linked to its usage rate. As the value of WGR increases, the number of coins needed to place a bet will decrease, and therefore, fewer coins will be destroyed. And alternatively, as the value of WGR decreases, the number of coins needed to place a bet will increase, and therefore, more coins will be destroyed. This change in rate of coin destruction is intended to balance out the value of WGR over time.

In Wagerr’s design, oracles are awarded a percentage of bettor fees and are incentivized to vote honestly to maintain their positions as Oracle Masternodes. A Masternode can be demoted due to poor performance, however. If an oracle often fails to vote with the consensus, has low network connectivity uptime, or has less than fifty percent participation in event submission in all supported leagues, they can be demoted.

Although Wagerr implements a more decentralized design, it still has many of the same drawbacks as ZenSports. Wagerr limits the events, on which users can place bets, to sporting events with only two potential outcomes. Additionally, Wagerr relies heavily on a relatively small set of oracles to both propose events as well as report outcomes of those events.

2.2.3 Augur

Augur (Figure 1c) is a decentralized platform for prediction markets that awards users for correctly predicting future events [10]. It is built on Ethereum, a cryptocurrency that supports smart contracts and, therefore, offers a platform on which other tokens can be built [27, 28].

Augur allows users to create markets representing specific events, and to purchase and sell “shares” of that market representing the potential outcome of those events. These “shares” act as wagers between two users. For example, consider a market for an event with two
outcomes, A and B. If Alice believes A will occur with a 70 percent chance, she can buy a share at a price of 0.7. If Bob believes B will occur with at least a 30 percent chance, he can purchase the remaining share for 0.3. As the event occurs and the outcome is reported, depending on which outcome occurred, either Alice or Bob will receive the entire share valued at 1.0.

Outcome reporting in Augur is initially done by a designated reporter referencing a designated “resolution source,” both specified by the market creator. The designated resolution source is meant to act as a trusted source that reporters use to determine the outcome of a market. Once the designated reporter has reported an outcome, the market enters a waiting period until the next fee window begins. The fee window is a 7-day time period during which reporting occurs. During this time, oracles can dispute the initially reported outcome by staking REP, Augur’s native token, on another outcome. When a consensus is reached among oracles (those users who hold Augur’s native token), the oracles who reported against the consensus will lose their stake and the oracles who reported with the consensus will receive a reward.

In order for Augur to operate and to incentivize honest reporting, it relies on two economies. The first is Ether (Ethereum’s native coin). Ether is used to purchase shares and cover the cost of gas charged by the Ethereum blockchain to execute Augur’s smart contracts. The use of Ether as the trading currency gives a large set of initial users (those users who hold Ether) the opportunity to participate. The second economy is REP (Augur’s native token). REP is used by oracles to stake votes on event outcomes and is awarded to oracles who vote with the consensus. The use of REP allows Augur to perform “Reputation Redistribution.” Reputation Redistribution is the act of paying out oracles who voted with the consensus any stake lost by oracles who voted against the consensus, after destroying twenty percent of any lost stake. With
this design, an oracle who successfully disputes an outcome is rewarded with a forty percent ROI on their initial stake.

Designating a single resolution source for reporters to determine a market’s outcome introduces a single source of failure to the system. Although there can be multiple reporters, if each reporter uses the same designated source as truth, the system becomes centralized and easily susceptible to attacks if a user can gain control of such a source. Augur also utilizes matchmaking, limiting the number of potential outcomes of an event to two. Additionally, because Augur allows oracles full control over their stake and the market on which they vote, it is susceptible to an attacker targeting a specific market with false votes.

2.2.4 Astraea

Astraea (Figure 1d) is a proposed decentralized platform to determine real-world event outcomes with a two-step voting process [6]. It is made up of three types of users: submitters, voters, and certifiers, and while the Astraea network is made up of human oracles, it is designed to be used by other blockchain applications as a standalone software oracle.

The voting process begins with submitters choosing which propositions will be voted on by submitting that proposition along with a small fee used to later reward voters and certifiers. Once a proposition is published, voters are allowed to vote on the outcome. The voter is required to first submit a stake, and only after submitting a stake can the voter vote on a randomly assigned proposition. Once a quorum is met, the outcome of the proposition is determined by the consensus vote, weighted by stake. At this point, certifiers are given the opportunity to “certify” a proposition’s selected outcome by staking their own vote. An important distinction between
voters and certifiers is that certifiers are allowed to select which proposition to certify before placing their stake, while voters are not. Finally, the “certified” outcome is chosen again, based on a consensus vote, weighted by stake.

If the outcome determined by majority vote for both voters and certifiers are equal, voters and certifiers who voted on that chosen outcome are rewarded while those that did not, lose their stake. If, however, the outcome determined by majority vote for both voters and certifiers are not equal, voters are returned their stake, certifiers lose their stake, and the outcome becomes “unknown.” It is important to note in the case where voters and certifiers disagree, Astraea makes no determination one way or the other on the outcome.

2.3 Deficiencies of Prior-Art

As detailed above, there are significant deficiencies and shortcomings with existing oracles. All oracles that we investigated only support events with two outcomes, while some place limits on supported event types as well as exhibit varying degrees of centralization. The challenge is how to resolve each of these deficiencies – building a platform that (i) puts no limitations on user proposed events including the number of potential outcomes; (ii) allows users themselves to report event outcomes in a trusted manner; (iii) provides a distributed betting protocol that puts no limitation on bet type and size; and (iv) removes all power from any centralized entity or entities.
2.4 Gambling Background

Due to the size and growth of the gaming industry, gambling offers an intriguing use case for a distributed oracle network. According to the American Gaming Association, an estimated $150 billion is wagered on sports each year [13]. And since the United States Supreme Court lifted the federal ban on sports betting in 2018 [29], twenty-six states have fully legalized sports betting and nineteen more have started the process of enacting similar legislation [30].

Casinos offer many benefits to a gambler. They create events on which to bet, they set lines and odds for those events, they determine winning outcomes, and they manage payouts to winning bettors. However, there are inherent deficiencies in centralized casino operations. Both physical and online casinos have bloated infrastructures with predominantly manual operations and processes. Brick and mortar casinos have large payroll structures, including dealers, bookmakers, and supplemental staff, that necessitate charging high fees to the players. These fees, typically masked as unfavorable odds, ensure that in the long run, the player will lose, and the casino will win (giving way to the notion that “the house always wins”).

There are two types of games in a casino – one played against the house and one played against another player. Games played against the house include slots, blackjack, and sports betting, where winnings are paid out by the casino itself, and games played against another player include poker and backgammon where winnings are paid out by the other player. Depending on the game, the casino’s fees are structured in different ways. For table games and sports betting, the fees are built into the payout structure of each bet and for games like poker, the casino implements a “rake” [31], a fee charged to players.

For instance, in American Roulette, a game with thirty-eight outcomes (1 through 36, 0,
and 00), the odds of a player winning a single number bet are 37 to 1. However, the odds set by casinos are not 37 to 1, as would be expected in a mathematically consistent and fair game. Instead, the rules of the game dictate that the payout for a single number bet is 35 to 1. This allows the “house to win.” In the long run, for every dollar bet on a single roulette spin, the casino will win approximately five cents [32].

This inherent advantage for the casino is also clear in non-table games, such as sports betting, due to the way that odds are set. A popular wager in sports betting is based on the “spread” [33]. The spread is essentially the number of points by which one team is favored over another. In these types of bets, the casino sets a spread based on public perception [34, 35] and to encourage half the bets on one team and half the bets on the other. A mathematically consistent and fair payout for the casino would be 1 to 1 (in other words, for each dollar bet, a dollar is won). However, this is rarely, if ever, the case. Casinos set the odds in such a way to give them up to a 10 percent advantage, requiring a player to bet as much as $1.20 to win $1.00.

Poker is one of the few games in a casino where the player does not “bet against the house.” In this case, however, a “rake” is established under which the casino charges the players to bet against each other. A rake can be collected in a number of ways, the most common being a pot rake or a time rake. A pot rake is where a portion of each pot is taken from the winner and paid back to the casino. A time rake is a fee the casino charges to each player for roughly every half hour of play. So, while players are, in fact, betting against one another, the casino is still charging a fee for the use of their tables, facilities, and dealers.

It is in these high fees and unfavorable odds that there exists an opportunity and need for a decentralized, user-minded, platform with the ability to provide players with a more affordable
and favorable means to engage in gambling.
CHAPTER 3: THE FORTUNA APPROACH

3.1 Introduction

Having identified the specific deficiencies of existing decentralized oracles, Fortuna seeks to resolve those deficiencies and shortcomings by establishing a fully distributed pari-mutuel gambling platform. All users will have the ability to propose events, place bets, accept payouts and, using a novel method of staked voting, determine event outcomes, without a centralized regulating entity. Moreover, as a pari-mutuel betting system [36], all bets are collected in a single pot, removing any restriction on the number of potential outcomes an event can have.

In contrast to the traditional “transaction” often used in other cryptocurrencies, Fortuna utilizes a more general “action” that encompasses many action types. An action can represent a pool, a bet, a vote, or a transaction. A pool represents a real-world event on which users can bet. A bet is a wager that a user places on the outcome of a pool. A vote is made to determine a pool’s outcome. Finally, a transaction is simply the transfer of coins from one user to another.

In a typical transaction-based cryptocurrency, each transaction requires a source address and a recipient address to ensure the network can verify both where the coin originated and where the coin was sent. This allows any user to verify the coin’s “owner” at any given time. In contrast, Fortuna pools, bets, and votes require only a source address to allow the network to verify the user performing the action. Because Fortuna pools represent real-world events, there is a necessary time period between pool creation, bet placement, and outcome determination. During this time period, the coins wagered on the pool are in a state of holding where there is no coin owner. Only after the pool has completed and the votes have been placed to establish the
outcome of the event, can a bettor redeem their winnings. By this methodology, Fortuna defines a coin as the chain of signatures between pools, bets, votes, and transactions, similar to how a typical transaction-based cryptocurrency defines a coin as the chain of signatures only between transactions.

3.2 Assumptions

For Fortuna to operate smoothly and successfully, a few assumptions must first be made. We first assume a set of expected attacks to include: (i) Sybil attacks; (ii) collusion; (iii) denial of service; and (iv) hash manipulation attacks. We also assume certain capabilities of attackers. We assume (i) attackers are willing and able to collude with other users by using communication outside of the platform (but not outright bribery); (ii) attackers have access to a large, but finite amount of capital; and (iii) attackers have incentive to attack the platform if they are able to earn more by lying than by being truthful. Finally, perhaps the most important assumption to be made is significant adoption of the protocol. As Fortuna relies on a cycle of betting and voting, as long as a consistent stream of pools are created and bets are placed, the network will be able to run smoothly and successfully.

3.3 Process Description

In this section, we will walk through a detailed description of the process users would experience when operating Fortuna (Figure 2). Much like oracles discussed in Section 2.2, Fortuna follows the same general oracle process. Below, however, is a more detailed algorithm showing how Fortuna differs from existing oracles.
Fortuna Oracle Process

1. Event proposed to the network
   a. Any user can propose a non-binary event.

2. Bets placed (optional)
   a. Any user can place a bet on an event’s outcome.

3. Event occurs in the real world

4. Oracles report outcome on blockchain
   a. Winning bettors of prior pools vote on this pool’s outcome.
   b. Quorum is reached and outcome with the majority vote is deemed the winning outcome.

5. Rewards/payouts occur
   a. Winning bettors are determined.
   b. Each winning bettor is assigned a range of pools on which they may vote.
   c. Each winning bettor votes on exactly one pool’s outcome.
   d. If a bettor votes with the majority, their winnings are paid out. If a bettor votes against the majority, their winnings are forfeited and paid out to other winning bettors.

The first step in the process is pool creation. Any user can create a pool and publish it to the Fortuna blockchain. There is a small fee of one coin to publish a pool to the blockchain. The purpose of this fee is to dissuade nefarious users from spamming the network with bogus pools. However, to cover the initial fee and to encourage users to create legitimate and appealing pools, the pool creator receives one percent (1%) of all bets on that pool unless the pool completes in a “no-contest” where the chosen outcome was not one specified by the pool’s creator. We empirically chose 1% based on existing fee structures. 0.5% is too small to incentivize pool creators, while casinos typically charge fees upwards of 5-10%.

In the pool, the user specifies a title, description, list of potential outcomes, and an expiration time for bets. The title and description are used to provide information to the bettors as to what event the pool represents. The list of potential outcomes is a list of outcomes for the
event on which users can place bets. The expiration time sets a deadline at which the pool stops accepting new bets and starts accepting votes. Finally, each pool is assigned a pool ID – a unique identifier generated by hashing the pool’s data. This pool ID is used later to place bets and votes on the pool.

As soon as the pool is published to the blockchain, it begins accepting bets. Any user can place a bet on any pool. To place a bet, the user specifies the unique pool ID of the pool on which the bet is placed, the outcome on which the bet is placed, and the amount of the bet. The outcome is an integer that corresponds to the index of the outcome in the pool’s list of outcomes. For example, in a pool representing the 2018 World Cup Final, the list of outcomes may be [“France Wins”, “Croatia Wins”, “Match Cancelled”]. If a bettor wished to place a bet on France winning, the bettor would set the bet outcome to 0. If a bettor wished to place a bet on Croatia winning, the bettor would set the bet outcome to 1. And, if a bettor wished to place a bet on the rare case of the World Cup Final being cancelled, the bettor would set the bet outcome to 2. Additionally, each bet is assigned a bet ID, a unique identifier generated by hashing the bet’s data. The bet ID of a winning bet is used later to place a vote on a pool.

Because Fortuna is a pari-mutuel betting system, the odds continue to change as more bets are placed. In order to hedge one’s bet, a bettor can place multiple bets on multiple outcomes of a pool. And, as noted previously, because pools represent real-world events, in the time between the placement of a bet and the determination of an outcome, these wagered coins are held in escrow and, therefore, cannot be utilized by any user on the network.

Once the expiration time for the pool is reached, the pool is closed. The pool stops accepting bets and starts accepting votes. To place a vote on a pool, a user will specify the
unique pool ID of the pool on which they wish to place their vote, the outcome of the event (i.e., their vote), and the bet ID of a previously won bet. Similar to a bet itself, the outcome is an integer corresponding to the index of the outcome in the pool’s list of outcomes. If the result of the event is not represented by an outcome in the pool’s list of outcomes, the voter sets the event outcome to \(-1\). The bet ID of a previously won bet is required in order to prevent users from placing an arbitrary number of votes on a given pool. Additionally, this requirement provides the voter with an incentive to vote honestly in order to redeem their winnings. It is important to note there is no designated “resolution source” for a given pool, but users are expected to do any necessary research before voting to ensure a successful vote. The pool will continue to accept votes until a quorum is reached. The methodology for establishing this quorum is discussed in Section 3.6.

Once quorum is reached, the pool is complete and the outcome with the majority vote is accepted as the winning outcome by the network. Users who placed bets on the winning outcome are now required to successfully vote on another pool in order to redeem their winnings. Successfully voting simply means voting with the consensus among other voters on that pool. Users who do not vote with the consensus will forfeit their winnings. These forfeited winnings will be treated as lost bets and paid out to the other winning bettors in the original pool. This procedure maintains voter honesty and network integrity.
Figure 2. Fortuna Process Description. Grey users denote bettors, voters, and transacting users. Purple users denote pool creators. The red user denotes Tyche (discussed in Section 4.3). And yellow circles denote coins. The first step in the process is pool creation. This is shown with the purple users. Each pool creator pays a 1-coin fee to create a pool, which is then collected by Tyche. The pool creator also collects 1% of all bets placed on a pool once the outcome has been determined (Pools 1, 2, 3, and 5). Note: Pool 4 has not been voted on yet, and therefore, the pool’s creator has not received their reward. Next, bettors place bets on their predicted outcome. However, Fortuna requires a winning bettor of one pool to vote on the outcome of another pool before redeeming their winnings. If a voter votes with the consensus, their winnings are released (Users 1, 2, 3, and 5), otherwise, their winnings are forfeited (User 4). And the process continues (Users 6 and 7). Losing bettors are not permitted to vote. Transactions can occur between any two users at any time. Three transactions are shown here (Users A, B, and C).
3.4 Staked Voting

As discussed in Section 2.2, a crucial requirement necessary of a decentralized network of oracles to ensure trust in its users is that each oracle must have an incentive to vote honestly or a disincentive to vote dishonestly. To satisfy this requirement, Fortuna implements a unique staked voting algorithm described below (Figure 3).

First, consider rounds of pools, bets, and votes. This is not a requirement in practice, but allows for easy illustration.

Let $p_i$, $b_i$, and $v_i$ define the set of pools, bets, and votes in the $i^{th}$ round. Round 1 begins with a set of pools $p_1$, bets $b_1$, and votes $v_1$. Now, assume the first round of pool outcomes have been determined and the network agrees. This concludes Round 1.

Round 2 begins in the same way, with a set of pools $p_2$ and bets $b_2$. At this point, the pools are closed, and the bettors are waiting on outcome determination from the voters $v_2$, to redeem their winnings.

Fortuna’s method of staked voting requires the winning bettors in Round 1 to vote on a random pool in Round 2 before redeeming their winnings from Round 1. Chosen randomly, the set of voters for the $j^{th}$ pool in Round 2, $v_{j2} \subseteq_R b_1' \subseteq_R b_1$ where $b_1'$ represents all winning bettors from Round 1. Additionally, a voter in the set $v_{j2}$ is only given access to their Round 1 winnings if they vote with the consensus among other voters in that set. This methodology is premised on the basic concept that the easiest way to ensure consensus among strangers is with the truth.

In practice, truly randomizing the set of voters for a given pool is difficult. However, we can approximate randomness by mathematically limiting the number of pools on which a user
can vote, based on a “voting partition.” We discuss voting partitions further in Section 3.5.

This method of staked voting satisfies the requirement defined above to minimize the feasibility of an attack on the integrity of the blockchain. Requiring each user in \( v_{j2} \) to vote with the consensus in order to redeem their Round 1 winnings incentivizes each voter to vote honestly.

Additionally, randomizing pool selection for a given voter defends against a Sybil attack [37]. A Sybil attack, in this scenario, would be one where a user could gain an advantage “controlling” a disproportionate percentage of the voting power simply by generating a large number of distinct accounts. In a voting system, this would allow a single user to place many votes on an event’s outcome with a relatively small upfront cost. Assigning random events on which users can place votes deters a user from attempting this type of attack. Controlling many accounts will not guarantee controlling many votes on a single event.

Using this staked voting method of outcome determination, each round of voting relies on the previous round of betting and, therefore, the network will run itself as long as subsequent pools are created and bets are placed.
Figure 3. Standard Staked Voting vs Fortuna Staked Voting. Grey users denote bettors and voters, and yellow circles denote coins. The left-hand side of the figure shows standard staked voting used in other decentralized gambling platforms. The users below the pool have placed bets on their predicted outcomes. Once bets are in, voters (users on top) stake coins on their vote for the outcome that actually occurred. If they vote with the consensus, they are rewarded, otherwise, they lose their stake. The right-hand side of the figure shows Fortuna’s method of staked voting. Fortuna requires a winning bettor of one pool to vote on the outcome of another pool in order to redeem their winnings. The figure shows bettors betting on three pools. Once the winning outcome is determined for Pools 1 and 2 (by voters not shown), the winning bettors now become voters. If a voter votes with the consensus, they redeem their winnings from their original Pool (Users 2 and 3), otherwise, they forfeit their winnings (User 1). Losing bettors are not permitted to vote.

3.5 Voting Partitions

As noted previously, in order for users to redeem their winnings from a winning bet, they must first vote with the consensus on a closed pool. However, to ensure random pool assignments and to defend against a Sybil attack, the pools on which a user can vote are
determined based on “voting partitions.” Voting partitions allow the number of pools on which a bettor can vote to be proportional to their bet amount – the smaller a user’s bet, the fewer pools we allow them to vote on – defending against an attacker placing a large number of small bets in the hopes of then placing a large number of dishonest votes on a particular pool.

A voting partition is defined by the leading $n$ bits of a pool ID. For example, if $n = 1$, there would be two partitions, those pools whose pool IDs begin with the bit $0$ and those that begin with the bit $1$. If $n = 2$, there would be four partitions, and so on. Using this design, the number of partitions is equal to $2^n$.

To determine the value of $n$, we use the following equation based on the size of the voter’s winning bet. As the size of the bet decreases, $n$ increases.

$$n = \max(1, 10 - \log_2(bet_{amount}))$$

Now that we have $n$, to determine the voting partition in which a user can vote, we use their voting value, $V$. The first $n$ bits of their voting value must equal the first $n$ bits of the pool ID of the pool they vote on. Below are three methods in which a voting value could be calculated, and possible attacks against them.

1. $V = bet_{id}$

One method would be setting a user’s voting value equal to their bet ID. At first, this appears safe. A user’s bet ID is unique to them and pseudo random. However, this is susceptible to a hash manipulation attack. Because the bet ID is the hash of a user’s bet, a user has the ability to alter their timestamp by small enough amounts to not be considered invalid, but to manipulate their bet ID in order to force the leading $n$ bits to equal whatever sequence they wanted. In this
way, the user would be able to vote on whichever pool or in whichever partition they wanted. To defend against this attack, we must introduce more randomness.

2. \[ V = bet_{id} \oplus vote_{id1} \oplus ... \oplus vote_{idn} \]

A second method would be setting a user’s voting value equal to their bet ID XORed with all vote IDs on their pool. Because these vote IDs are essentially random to the bettor at the time they place their bet, they would not be able to manipulate their bet ID in a way to target a specific pool. However, this design is still susceptible to a hash manipulation attack in a different way. Because \( vote_{id1} \oplus ... \oplus vote_{idn} \) does not change for a given pool, that value can be considered constant. Therefore, a user could place many small bets on a pool and manipulate their bet IDs such that the leading \( n \) bits were all equal, regardless of what the specific sequence was. In this way, when their bet IDs are XORed with these vote IDs, although a user could not force the leading \( n \) bits to equal a specific sequence, they could force the leading \( n \) bits to all be equal. Therefore, while a user would not be able to target a specific partition with this design, they would still be able to target a partition.

3. \[ V = \text{hash}(bet_{id} \oplus vote_{id1} \oplus ... \oplus vote_{idn}) \]

The final solution to defend against both of these attacks is to take the hash of Eq. 2. Now, in the same way the pool ID is used to assign a voting partition to a pool, this value \( V \) is used to assign a voting partition to a voter, as shown in Figure 4.
3.6 Quorum

Quorum is used to determine when voting on a pool is closed and payouts can be made. How this quorum is defined is crucial in ensuring the network runs smoothly and efficiently. If the quorum is too small, there will be a bottleneck of bet winners waiting to vote. If the quorum is too large, there will not be enough winning bettors to satisfy this quorum requirement and pools will fail to complete in a timely manner. At any given time, the total quorum across all closed pools must equal, on average, the number of voters, or winning bettors. Additionally, the quorum should grow as the network’s user base grows. As the number of pools and bets
increases, the size of the required quorum should also increase.

On average, an outcome of a given pool will have \( \frac{|bets|}{|outcomes|} \) bets placed on it, where \( |bets| \) is the number of bets placed on a pool and \( |outcomes| \) is the number of potential outcomes on that pool. Therefore, the expected number of winning bettors from a given pool is the same, because there is only one winning outcome. To ensure the total quorum across all pools does not diverge from the number of winning bettors, we define the quorum for a given pool to be:

\[
Quorum = \frac{|bets|}{|outcomes|}
\]

3.7 Payout

As noted above, once the quorum of a pool is reached and a winning bettor has voted with the consensus on another pool, they will be allowed to redeem their winnings. The winnings of a bet are determined by the ratio of total losing bets to total winning bets (minus one percent to reward the pool’s creator). Let \( W \) be the winnings from a bet, \( b \) be the amount of the bet, \( b_l \) be the set of all losing bets on that pool, and \( b_w \) be the set of all winning bets on that pool:

\[
W = 0.99 \cdot b \cdot \left( 1 + \frac{\sum b_l}{\sum b_w} \right)
\]

For example (Figure 5), consider three bets placed on the 2018 World Cup Final. Alice bet 20 coins on France to win. Bob bet 40 coins on France to win. And Carol bet 90 coins on Croatia to win. After voting establishes that France was the winning outcome and after both Alice and Bob have voted with the consensus on other pools, Alice will be awarded 49.5 coins
(50 minus one percent awarded to the pool’s creator) and Bob will be awarded 99 coins (100 minus one percent awarded to the pool’s creator). If, however, Bob votes against the consensus on another pool, his winnings (99 coins) will be distributed to the other winning bettors in the original pool, in this case, Alice, weighted based on their bet amounts. Alice’s new winnings would be 148.5 coins.

**Figure 5. Winning Bettor Payouts.** Grey users denote bettors, purple users denote pool creators, and yellow circles denote coins. The left-hand side of the figure shows the scenario where Bob voted with the consensus. In this scenario, the payout is determined by splitting Carol’s lost bet of 90 coins between all winning bettors (Alice and Bob) based on the amount of their bets, after paying the pool creator 1% of all bets (150 total coins bet; 1.5 coins paid to pool creator). The right-hand side of the figure shows the scenario where Bob voted against the consensus. In this scenario, the payout is determined by splitting Carol’s lost bet of 90 coins and Bob’s forfeited bet of 40 coins between all other winning bettors (Alice) based on the amount of their bets, after paying the pool creator 1% of all bets (150 total coins bet; 1.5 coins paid to pool creator).
CHAPTER 4: IMPLEMENTATION

This chapter outlines Fortuna implementation details including balances, fees, the bootstrapping process, and its protocol (Figure 6) – supported action types and their components, as well as validation requirements for each.

4.1 Balances

In the Fortuna protocol, actions themselves hold balances and provide proof of funds. At any point in time, the balance of an action can be calculated. For example, to calculate the current balance of Action A, one can step through the blockchain calculating the total cost of each action referencing Action A and subtracting that cost from its initial balance.

The initial balance of an action is calculated based on the action type. The initial balance of a pool depends on the outcome of the pool. If the outcome selected by a majority vote was an outcome identified in the pool’s list of outcomes, the pool’s creator will be awarded one percent of the total pot. Thus, the initial balance of that pool would be one percent of all bets placed on that pool. However, if the majority outcome is a “no-contest,” where the pool’s outcome was not provided in the list of outcomes, the pool’s creator will receive no reward. Thus, the initial balance of a “no-contest” pool is zero.

Like a pool’s initial balance, the initial balance of a vote depends on the outcome of the vote. If the user votes with the majority, the vote’s initial balance will be equal to the winnings of the previously won bet referenced in the vote. If the user votes against the consensus, the initial balance of the vote will be zero; thus, winnings from the previously won bet are forfeited.

And lastly, a transaction’s initial balance is simply the amount of the transaction sent to
that user.

4.2 Fees

Fortuna is built on its own custom blockchain and, therefore, has the luxury and benefit to users of charging minimal fees. The fees it does charge are necessary, however, to maintain the network and ensure the integrity of system operation.

The first fee is charged to any user to create a pool. There is a fee of one coin required to publish a pool to the blockchain intended to dissuade nefarious users from spamming the network with bogus pools. This fee is collected by the Tyche address (discussed in Section 4.3). However, to reward users for creating pools and to encourage creation of popular and competitive pools, once the pool closes, if the consensus outcome was one provided in the list of outcomes, the pool’s creator will be awarded one percent of the total bets placed on that pool.

The second fee is charged to place a bet. There is a one percent fee on all bets. This fee is used to reward the pool’s creator and is the only fee that bettors will be required to pay. If a pool ends in a “no-contest” though, bettors will be returned their full bet amount including the one percent fee. Because fees in casinos and other centralized gambling applications can be as high as ten percent, this one percent fee paid out to pool creators is reasonable.

4.3 Tyche

Fortuna incorporates a master address – Tyche. This master address is used for two primary reasons. First, this address is used to “bootstrap” the system – to fund all initial investor addresses in the genesis block (as detailed in Section 4.4). Second, this master address is used to
collect all pool creation fees providing compensation to the Fortuna creator.

4.4 Bootstrapping and the Genesis Block

The Fortuna system is dependent on a cycle of winning bettors voting on the outcome of future pools. Accordingly, there is a systemic “gap” as to the initial pool. For the first pool created, and the first bets placed, there will be no previous set of bet winners to vote on the outcome. The solution is Fortuna’s “bootstrapping” methodology.

The genesis block is the first block in the blockchain and is used to validate all future actions. Additionally, the genesis block will be used to bootstrap the Fortuna protocol using initial investors' addresses and the Tyche address. The genesis block is not validated in the same manner as subsequent blocks, but must be accepted as truth.

The genesis block will contain four sets of actions. First, it will contain a transaction from Tyche to the investors in the amount of 50 percent of their initial investments. Second, it will contain a pool, the genesis pool, created by Tyche with a single outcome. Third, it will contain bets on the genesis pool that Tyche makes on behalf of the investors using the remaining 50 percent of an investor’s due funds. Each investor must sign their bet when they invest in Fortuna. Lastly, the genesis block will contain a vote from Tyche (deemed sufficient to form a quorum only in the genesis block) on the pool's single outcome.

Once Tyche publishes the genesis block, the Fortuna protocol is fully “bootstrapped.” If a user wishes to create a pool, place a bet, or execute a transaction, that user can reference the action ID of the initial transaction originating from Tyche in the genesis block. And if a user wishes to vote on a pool, they can reference their winning genesis bet allowing the user to
redeem the remaining 50 percent of their initial investment.

It is important to note the initial investment made by investors will not make up the entirety of the Fortuna economy. The genesis block and initial investment is necessary only to bootstrap the process and cycle of betting and voting. New users wishing to participate would be able to earn coins by mining new blocks of actions or by purchasing coins from a cryptocurrency exchange.

4.5 Action

Actions (Figure 6) are used to interact with the Fortuna network and can represent pools, bets, votes, or transactions.

![Figure 6. Class Diagram of Fortuna’s Action Protocol](image-url)
All actions are comprised of a header and a payload. The header contains two fields: action_id and signature. The action_id is a unique identifier of the action and is created by hashing the payload data and is used later by other actions for validation. The action_id can be referred to as the pool_id, bet_id, vote_id, or transaction_id, depending on the specific action type. The signature is the signature of the payload generated by the user’s private key. Finally, the payload is a base class representing either a Pool, Bet, Vote, or Transaction. The components of an Action are detailed below (Table 1).

### Table 1. Class Parameters for an Action

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Data Type</th>
<th>Field Size</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>action_id</td>
<td>byte[32]</td>
<td>32</td>
<td>Unique identifier of the action</td>
</tr>
<tr>
<td>signature</td>
<td>byte[64]</td>
<td>64</td>
<td>The signature of the payload</td>
</tr>
<tr>
<td>payload</td>
<td>Payload</td>
<td>?</td>
<td>Pool/Bet/Vote/Transaction</td>
</tr>
</tbody>
</table>

4.5.1 Payload

All action payloads contain five fields: action_type, public_key, address, reference_id, and timestamp. The payload also includes additional data specific to the action type. The action_type denotes whether this action is a Pool, Bet, Vote, or Transaction. The public_key is an Ed25519 public key provided by the user to validate the signature. The address is a unique identifier for the user and is generated by hashing the public key. The reference_id is the action ID of a previously validated and published action and is used to validate this action. For example, if this action represented a pool, bet, or transaction, the reference ID would need to correspond to an action in which this user received enough coins to cover the cost of this new
action. If, however, this action represented a vote, the reference ID would need to correspond to a previously won bet. Finally, the timestamp is the Unix time at the time of action creation. The components of an action payload are detailed below (Table 2, Table 3).

**Table 2. Class Parameters for an Action Payload**

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Data Type</th>
<th>Field Size</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>action_type</td>
<td>byte</td>
<td>1</td>
<td>Denotes the type of Action</td>
</tr>
<tr>
<td>public_key</td>
<td>byte[32]</td>
<td>32</td>
<td>User’s public key</td>
</tr>
<tr>
<td>address</td>
<td>byte[32]</td>
<td>32</td>
<td>Unique user identifier</td>
</tr>
<tr>
<td>reference_id</td>
<td>byte[32]</td>
<td>32</td>
<td>Reference action_id</td>
</tr>
<tr>
<td>timestamp</td>
<td>int</td>
<td>8</td>
<td>Timestamp of action creation</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>Payload data</td>
</tr>
</tbody>
</table>

**Table 3. Action Types and Corresponding Values**

<table>
<thead>
<tr>
<th>Action Type</th>
<th>Description</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Description</td>
<td>Value</td>
</tr>
<tr>
<td>Pool</td>
<td></td>
<td>0x01</td>
</tr>
<tr>
<td>Bet</td>
<td></td>
<td>0x02</td>
</tr>
<tr>
<td>Vote</td>
<td></td>
<td>0x03</td>
</tr>
<tr>
<td>Transaction</td>
<td></td>
<td>0x04</td>
</tr>
</tbody>
</table>

4.5.2 Pool

A pool contains four additional fields: title, description, outcomes, and expiration_time. The title is the title of the pool’s event (e.g., “2018 World Cup Final”) and is limited to 64
characters. The *description* is simply a brief description of the pool (e.g., “This pool represents the 2018 World Cup Final between France and Croatia played on 15 July 2018.”) and is limited to 256 characters. *Outcomes* is a list of possible outcomes on which bettors can place bets. Each outcome is a string representing a specific outcome (e.g., [“France Wins”, “Croatia Wins”, “Match Cancelled”]) and is limited to 64 characters. Finally, the *expiration_time* is the Unix time at which the pool closes to new bets and starts accepting votes. The components of a Pool are detailed below (Table 4).

**Table 4. Class Parameters for a Pool**

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Data Type</th>
<th>Field Size</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>title</td>
<td>var_str(64)</td>
<td>?</td>
<td>Title of pool</td>
</tr>
<tr>
<td>description</td>
<td>var_str(256)</td>
<td>?</td>
<td>Description of pool</td>
</tr>
<tr>
<td>outcomes</td>
<td>var_str(64)[?]</td>
<td>?</td>
<td>List of possible outcomes</td>
</tr>
<tr>
<td>expiration_time</td>
<td>int</td>
<td>8</td>
<td>Expiration time</td>
</tr>
</tbody>
</table>

### 4.5.3 Bet

A bet contains three additional fields: *pool_id*, *outcome*, and *amount*. The *pool_id* is the action ID associated with the pool on which this bet is placed. The *outcome* is an integer corresponding to the index of the outcome in the pool’s list of outcomes on which this bet is placed. Finally, the *amount* is the number of coins wagered on this bet. The components of a Bet are detailed below (Table 5).
<table>
<thead>
<tr>
<th>Field Name</th>
<th>Data Type</th>
<th>Field Size</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pool_id</td>
<td>byte[32]</td>
<td>32</td>
<td>The pool ID on which this bet is placed</td>
</tr>
<tr>
<td>outcome</td>
<td>int</td>
<td>8</td>
<td>The chosen outcome</td>
</tr>
<tr>
<td>amount</td>
<td>int</td>
<td>8</td>
<td>Amount to wager</td>
</tr>
</tbody>
</table>

4.5.4 Vote

A vote contains two additional fields: pool_id and outcome. The pool_id is the action ID associated with the pool on which this vote is placed. The outcome is an integer corresponding to the index of the outcome in the pool’s list of outcomes on which this vote is placed. The components of a Vote are detailed below (Table 6).

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Data Type</th>
<th>Field Size</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pool_id</td>
<td>byte[32]</td>
<td>32</td>
<td>The pool ID on which this vote is being placed</td>
</tr>
<tr>
<td>outcome</td>
<td>int</td>
<td>8</td>
<td>The chosen outcome</td>
</tr>
</tbody>
</table>

4.5.5 Transaction

A transaction contains two additional fields: recipient_address and amount. The recipient_address is the address associated with the user to which this transaction is being sent. The amount is the number of coins being sent as part of this transaction. The components of a Transaction are detailed below (Table 7).
Table 7. Class Parameters for a Transaction

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Data Type</th>
<th>Field Size</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>recipient_address</td>
<td>byte[32]</td>
<td>32</td>
<td>Address of recipient</td>
</tr>
<tr>
<td>amount</td>
<td>int</td>
<td>8</td>
<td>Amount to send</td>
</tr>
</tbody>
</table>

4.6 Validation

Validation is performed on every new action and block added to the blockchain. This step is crucial to maintain a consistent and trusted blockchain between otherwise trustless users. Validation ensures that users do not spend coins they do not own, place bets on closed pools, or nefariously vote on pools without having previously won a bet. Full nodes are responsible for validation and, like Bitcoin full nodes [38, 39], are volunteers keeping the Fortuna network running smoothly and honestly.

The use of reference IDs are crucial in validation. A reference ID is the unique action ID of a previously validated action that a user must reference in order to provide either proof of funds or proof of bet for a new action. Proof of funds is required when taking any action that requires coins, such as creating a pool, placing a bet, or making a transaction. Proof of bet is required when placing a vote, as the user must have previously won a bet in order to vote on a pool.

To provide proof of funds, reference IDs can refer to pools, votes, or transactions. These action IDs hold balances used to ensure users are not placing a bet or sending a transaction that exceeds their current balance. For example, if a user wishes to place a bet for 10 coins, that user must provide a reference ID of an action in which they received at least 10 coins. A user can
reference the pool ID of a pool which they created, having received a reward for creating that pool. A user can reference the vote ID of a vote, thereby redeeming winnings from a previously won bet. Or finally, a user can reference the transaction ID of a transaction in which they were sent coins from another user. To provide proof of bet, reference IDs can only refer to previously won bets.

4.6.1 Actions

All actions include a header and a payload. The header contains two fields – the action ID and a signature. The action ID, sometimes referred to as the pool ID, bet ID, vote ID, or transaction ID, is a unique identifier of the action and is created by hashing the payload data. The signature is the signature of the payload data generated by the user’s private key. The payload holds the remaining data, including the action type, the user’s public key, the user’s address, the reference ID, and the timestamp of the action, along with additional data specific to the pool, bet, vote, or transaction.

There are five initial validation requirements for all actions. First, the action ID is required to be the SHA-256 hash of the payload. Second, the address provided in the action is required to be the SHA-256 hash of the public key. Third, the signature is validated using the user’s public key. The Ed25519 signing scheme is used to keep key size and signature size small. Fourth, the previous action, corresponding to the reference ID provided in the action, exists in the blockchain; this validation step assumes that, if an action has been published to the blockchain, it is, itself, valid. Lastly, the timestamp provided in the action differs by no more than one minute of the validator’s system time.
This one-minute grace period is required to allow the action to propagate throughout the network. This was chosen based on previous work in which it was determined that it takes approximately 40 seconds for a Bitcoin transaction to propagate to 95 percent of the network [38]. Setting this grace period to one minute ensures a large majority of nodes are able to successfully validate the message.

It is important to note that while pools, votes, and transactions do not necessarily require a “valid” timestamp, bets do. A nefarious user may attempt to place a bet on a pool by waiting until the end of the event and backdating the timestamp in order to falsely place a bet on an already completed pool. The requirement of a “valid” timestamp on bets makes this type of attack less affective.

Users can still attempt to attack the system, however. For example, given a pool with an expiration time of May 4, 6:30:00, a bettor could place a bet thirty seconds after expiration, at 6:30:30, but backdate their timestamp to 6:29:59. A validator would first compare the bet’s timestamp to the pool’s expiration and then to their own system time, say 6:30:45, and approve it. So even with only a one-minute grace period, there is still margin for an attacker to place a bet after a pool expires. Ideally, a pool creator would set the expiration a few minutes prior to the start of the event. But, since there is no mechanism preventing a pool creator from setting the expiration to even after an event is scheduled to end, some responsibility is put on the bettor to judge the validity of a pool and make smart bets.

4.6.2 Pool

In addition to the action validation requirements described above, pool validation requires
that the current balance of the action, corresponding to the reference ID provided in the pool, be no smaller than the fee (1 coin) associated with creating a pool.

4.6.3 Bet

In addition to the action validation requirements described above, bet validation requires the following:

1. The pool on which the bet is placed, corresponding to the pool ID provided in the bet, must exist in the blockchain;
2. The timestamp of the bet must be no later than the expiration time of the pool on which the bet is being placed; and
3. The current balance of the action, corresponding to the reference ID provided in the bet, must be no smaller than the size of the bet.

4.6.4 Vote

In addition to the action validation requirements described above, vote validation requires the following:

1. The pool on which this vote is placed, corresponding to the pool ID provided in the vote:
   a. Must exist in the blockchain.
   b. Has not yet reached a quorum.
   c. Must belong to the correct voting partition.
2. The bet, corresponding to the reference ID provided in the vote:
   a. Exists in the blockchain.
b. Was authored by the same address associated with this vote.

c. Is a winning bet.

d. Has not been referenced in another vote.

4.6.5 Transaction

In addition to the action validation requirements described above, transaction validation requires that the current balance of the action, corresponding to the reference ID provided in this transaction, be no smaller than the size of the transaction.
CHAPTER 5: RESULTS AND DISCUSSION

5.1 Results

To test the described protocol above, a simulator was constructed, and simulations were run of sets of users interacting with the network. After the genesis block was mined, these users were responsible for creating new actions, validating new actions, mining blocks, and managing balances and payouts over a period of time. The network was made up of one hundred simulated users, each with an initial investment of one thousand coins and the network ran for one hundred blocks.

To begin the simulation, a genesis block was mined by the Tyche address. This block included one hundred transactions (sending each user half of their initial investment), a pool with a single outcome, a bet made by each user of their remaining investment and, finally, a vote made by the Tyche address on the genesis pool. This fully bootstrapped the system.

At this point, each user generated a random set of actions to be validated and published to each block. These actions were taken based on probabilities incorporated into the simulator. For example, the probability that a user would create a pool was 20%. The probability that a user would place a bet on a randomly chosen pool was 80%. The probability that a user would cast a vote, given they had a previously won bet, was 100% and the probability that they would vote erroneously or maliciously was 5%. Finally, the probability that a user would send a transaction to another randomly chosen user was 20%. These probabilities were chosen after trial and error to allow a large number of bets to be placed on each pool while also incorporating a non-negligible amount of erroneous or malicious votes. After validation, this set of actions was reduced to a smaller set of validated actions and included in the next block, which was then
mined and published to the blockchain by a random user (awarding that user with their mining reward). This process continued until the last block was mined.

To test the performance of the blockchain and the protocol, the expected total value of the network was compared to the actual total value of the network at the end of the last block in the simulation. The expected value was calculated as the initial balance of all users in addition to the expected mining reward for each new block. The actual value was calculated with the final balance of all user actions at the end of the simulation, in addition to the value of all open bets (those bets placed on a pool that had not yet closed), and all unredeemed bet winnings (those winnings that had not yet been paid out with a successful vote). While the coins in the network move between users, pools, bets, votes, and transactions, the total value should stay consistent. The simulation demonstrated that expected consistency.

5.2 Discussion

5.2.1 51% Attack

In cryptocurrencies, a 51% attack is an attack a nefarious user can make by owning over half of the mining power on the network. Owning over half of the mining power would allow an entity to mine blocks faster than the rest of the network. Because cryptocurrencies define the longest chain as truth, an ability to mine blocks faster than the rest of the network would allow a nefarious user to create the longest chain (thus establishing the “truth”), in a unilaterally determined manner. Such an attack could prevent certain users from placing transactions altogether or “erasing” previous transactions from the network, allowing a double spend. For example, if an attacker makes a transaction, cashing out $100, the network would see that
transaction and validate it. Then, outside of the network, a cryptocurrency exchange would send the attacker $100. All the while, the attacker could secretly mine their own blocks without that transaction included. Once the attacker received the $100 offline, the secret chain could be published to the network. Because the secret chain would be longer than the current longest chain, the network would treat that chain as the truth. In that chain though, the attacker has not cashed out the $100 and, therefore, could spend those coins again – a “double spend.”

While Fortuna is still theoretically susceptible to a 51% attack in the double spend scenario, the voting protocol is defensive due to its staked nature. An attacker is not able to arbitrarily place votes on a pool to ensure a false outcome even with over half of the mining power. Because voters are required to have previously won a bet, and bet winners are effectively randomly assigned pools on which they can vote, an attacker would need to stake a large amount of money on a large number of bets in order to ensure having a majority vote on any single pool. The required investment should dissuade users from attempting to carry out such an attack.

5.2.2 Open Voting

Fortuna uses open voting. Open voting allows users to see the votes of other users before placing their own votes. For this reason, an interesting scenario could arise. Consider a scenario in which Alice has just won a bet and now needs to vote with the consensus on another pool in order to redeem her winnings. She wishes to vote on the 2018 World Cup Final and, having watched the match, she knows France beat Croatia. However, she sees that some votes have already been placed – all for Croatia. In this situation, Alice might be discouraged to vote honestly and instead vote dishonestly with the majority, in order to redeem her winning bet.
The solution to this would be implementing some sort of closed voting, where a user could not see the votes until a quorum is reached. A possible solution we considered was using a “commitment scheme.” A commitment scheme is a cryptographic scheme in which a user can commit to a certain value without revealing that value until later. A basic commitment scheme could be implemented here using a hash and a “salt.” A salt is a secret value provided to a hash, similar to a private key, in order to add randomness to the hashed value. For example, Alice, wishing to place a vote on the 2018 World Cup Final, can either vote for “France” or “Croatia.” Instead of placing an open vote for “France” that any user on the network could see, Alice will instead calculate and broadcast a hash, \( V = \text{hash}("France", b) \), where \( V \) is her commitment and \( b \) is the randomly generated salt she uses to create the hash. Once the pool has reached a quorum of voters, Alice can then broadcast \( b \) to the network, “revealing” her vote.

While this solves the initial problem of a few bad actors voting dishonestly at the beginning of a pool’s voting process and thereby possibly affecting future votes, it introduces a new vulnerability. A single nefarious or absent-minded user may broadcast their commitment to vote, but then later fail to broadcast their salt, leaving the pool in limbo, waiting for all votes to be released. There must be a disincentive for a user to fail to release their salt. A possible solution is to treat a hanging vote as a dishonest vote, forfeiting the user’s winning bet. This though, leads to the question of “how long is too long for a vote to hang?” Additionally, this solution requires a second action for a winning bettor to redeem their winnings. This is an area for further study.

At this time, the added complication to the user experience and protocol was not worth the added defense against what might not be a problem in practice. If, in fact, this scenario arises,
would a user be willing to bet their winnings that the remaining voters on a pool would continue

to vote dishonestly? Moreover, the voter can provide the easiest solution – if there is a pool that

is unappealing for a user to vote on (i.e., one in which certain dishonest votes have been cast),

the voter can choose to vote on another pool in their voting partition. As described in Section 3.5,

the user is not limited to a single pool on which to vote.

5.2.3 Staked Voting Attack

As with any decentralized network, there are attacks available to nefarious users, and

Fortuna is no different. It can be shown that with enough stake, a nefarious user could potentially

attack Fortuna’s voting protocol and sway the consented outcome of a pool. However, with its
design, this attack is much more difficult and expensive to carry out than in existing protocols.

Assume a pool has only two potential outcomes and $B$ total bets. The quorum would

equal $\frac{B}{2}$ and therefore, an attacker would need to control at least $\frac{B}{4}$ votes to create a majority on a

chosen outcome. Because Fortuna uses voting partitions to determine the set of pools on which a

user can place votes, an attacker would need to guarantee their votes all fall within a single

partition. As discussed in Section 3.5, voting partitions are determined based on a user’s bet size.

A bet of 512 coins would result in a partition covering $\frac{1}{2}$ of all pools where a bet of 1 coin would

result in a partition covering $\frac{1}{1024}$ of all pools. Therefore, with $\frac{B}{2}$ winning bets of 512 coins each,

an attacker is guaranteed at least $\frac{B}{4}$ votes in a single partition, allowing themselves to control the

outcome of a pool in that partition.

51
With a required initial investment of $256 \times B$, as the popularity of a pool increases, so does the difficulty and required initial investment to attack it. Additionally, the potential winnings for an attacker are limited. Because Fortuna is a pari-mutuel system, winnings are limited by the amount bet on all outcomes. An attacker would not be able to make an arbitrarily large bet to ensure an arbitrarily large payout. And lastly, even with a large number of votes, an attacker is not guaranteed the ability to control a pool’s outcome as miners mine actions as they come in. If enough honest votes are placed before all dishonest votes are mined, the pool’s quorum will be met and voting will close. The attacker could potentially fall short of the majority, resulting in losing their bet as well as forfeiting all of their winnings from their previously won bets.

Augur is the only other gambling platform with a truly decentralized voting protocol but has a much simpler attack. As opposed to Fortuna, Augur sets no requirement on a user prior to their staking a vote. Any user can vote on any market. In Augur’s protocol, if an attacker simply stakes more REP on their votes than the other voters, they can decide the outcome of a market. The only security Augur implements is allowing a certain amount of time for users to override a selected outcome after one has been chosen, but there is no guarantee this will happen, and, in fact, can be very difficult with a large enough initial stake.

While Fortuna’s staked voting protocol is still susceptible to an attack, we believe the large required initial investment due to the use of voting partitions and requirements placed on voters, creates a more robust protocol than existing platforms and the risk associated with attempting to control a pool makes this attack impractical.
CHAPTER 6: CONCLUSION AND FUTURE WORK

6.1 Summary and Conclusion

The problem that this thesis set out to solve is in how to bring real-world event outcomes onto a blockchain in a trusted way. We solve this problem by introducing Fortuna, a distributed oracle network demonstrating a novel staked voting algorithm, introducing the concept of voting partitions, and implementing a unique pari-mutuel betting design.

As impactful as decentralized blockchain technology has become since the inception of Bitcoin in 2008, there are still challenges. One challenge addressed in this thesis is the ability to bring real-world event outcomes onto the blockchain. Most decentralized blockchains do not have this capability. However, there are existing oracle networks that attempt to solve this problem. As discussed in Section 2.2, previous attempts to create such a network include ZenSports, Wagerr, Augur, and Astraea, each of which includes drawbacks such as: (i) major limitations on supported event types; (ii) reporting algorithms susceptible to attacks; and (iii) varying degrees of centralization.

Centralized casinos offer many advantages and benefits to gamblers and, while there are obvious differences between casinos and decentralized oracle platforms, they share many deficiencies from the bettor’s perspective – high costs, lack of betting flexibility, and varying degrees of centralized authority. Nevertheless, an estimated $150 billion is bet on sports each year in the United States [13]. This centralized industry is the perfect application for a decentralized oracle network.

Fortuna, a distributed gambling platform, motivated both by the large amount of capital flowing through the gambling industry as well as the need for a more robust and capable oracle
platform, solves these problems in a novel way. Fortuna is designed to provide players with a more affordable and favorable means to engage in gambling through its use of actions, representing pools, bets, votes, and transactions. Users interact with the network by creating pools representing real-world events, placing bets on those pools, casting votes on pool outcomes, and transacting with other users.

In order to incentivize voters to vote honestly on event outcomes, Fortuna implements a novel staked voting algorithm. This is achieved by requiring winning bettors to vote with the consensus on another pool in order to redeem their winnings. If the user votes with the consensus, their winnings will be released. If, however, the user votes against the consensus, their winnings will be forfeited and paid out to the other winning bettors.

As winning bettors are required to place votes in order to redeem their winnings, to defend against a Sybil attack and ensure random pool assignment, Fortuna introduces the concept of voting partitions. Voting partitions allow the number of pools a bettor can vote on to be proportional to their bet amount – the smaller a user’s bet, the fewer pools the bettor may choose to vote on. This defends against an attacker placing a large number of small bets in the hopes of then placing a large number of dishonest votes on a particular pool.

Additionally, because there is no centralized casino against which a user can place a bet, bets are made with, and against, other users. A pool accepts bets from all sides and pays out winning bettors based on the total pot. This pari-mutuel design allows the odds of a bet to change organically based on public perception and, with enough bets, approach an equilibrium. In addition, this removes any requirement for a binary outcome. With a pari-mutuel design, there is effectively no limit on the number of potential outcomes an event can have.
This method of staked voting, the concept of voting partitions, as well as the pari-mutuel design creates a fully decentralized, self-handicapping, and self-regulating gambling network and allows non-binary real-world event outcomes to be determined in a reliable and trusted way by otherwise trustless users.

6.2 Future Work

While the system and protocol have been tested with favorable results, there are areas of potential improvement and further study.

One opportunity for improvement is the ability to reference multiple previous actions in a new action. As discussed in Section 4.1, a new action requires a reference ID of a previously taken action in order to provide proof of funds. A user wishing to create a transaction (e.g., sending 10 coins to another user), is required to reference a previous action (where the user received 10 coins). This allows miners and other users to validate new actions. However, if a user was sent two transactions of 5 coins each and that same user wished to send a transaction of 10 coins to another user, it would not be possible to reference a single action in order to provide proof of funds. The user would need to reference multiple previous actions. Allowing a user to reference multiple previous actions in support of a single new action is essential for this cryptocurrency to be adopted and successful.

Another opportunity for improvement is the implementation of a solution to the possible problem stemming from open voting, discussed in Section 5.2.2. Open voting allows a voter to see all previous votes on a pool before making their own. As noted previously, if the initial votes placed on a pool are dishonest, a new voter may be inclined to vote dishonestly as well if that
appears necessary to vote with the majority. Consideration was given to using a commitment scheme, keeping votes secret until a quorum is reached, but it was decided not to proceed with implementation due to other complications and drawbacks, such as a voter failing to reveal their commitment leading to a hanging pool, as well as the additional complication placed on the user in order to redeem their bet winnings. Future work should focus on development of an elegant means of implementing a commitment scheme or otherwise mitigating this potential problem.
REFERENCES


