Distributed Dependable Systems

Bachelor Thesis

Design And Implementation Of A Carbon-Asset Backed Lending Protocol With Vote-Escrow Tokenomics To Achieve Sustainable Liquidity And User Engagement

Lilly Guo

Matr. 123456

First Reviewer: Prof. Dr. Katinka Wolter
Second Reviewer: MSc. Justus Purat

Institute of Computer Science, Freie Universität Berlin, Germany

October 21, 2023

Declaration of independence

I hereby declare to have written this thesis on my own. I have used no other literature and resources than the ones referenced. All text passages that are literal or logical copies from other publications have been marked accordingly. All figures and pictures have been created by me or their sources are referenced accordingly. This thesis has not been submitted in the same or a similar version to any other examination board.

Berlin, October 21, 2023

(Lilly Guo)

Abstract

Abstract

This thesis presents a platform that can increase the adoption of carbon tokens in DeFi, developing a decentralized lending and borrowing protocol that supports carbon tokens as collateral. It implements a novel tokenomics system based on staking tokens into a vote-escrow contract to gain voting power to direct token emissions to favored pools. This system incentivizes liquidity provision and encourages long-term user engagement with the protocol. This thesis presents a proof of concept of lending and borrowing with carbon tokens and a rigorously designed and thoroughly tested vote-escrow contract.

Contents

CHAPTER 1

Introduction

This chapter introduces the domain of decentralized finance applications, cryptocurrency assets and tokenization, and carbon certificates. It describes the motivation for bringing carbon certificates onto the blockchain, and presents the platform designed in this thesis to increase the adoption and usage of carbon tokens in decentralized finance.

1.1 Motivation

Decentralized finance ("DeFi") applications are programs in the form of smart contracts that run on a decentralized blockchain network. They enable individuals to access financial services directly on the blockchain ("on-chain") without the need for intermediaries such as banks or brokers. This paradigm shift has opened up new possibilities for secure, transparent, and efficient financial transactions.

In recent years, the tokenization of real-world assets has gained significant traction in the blockchain space. This process involves representing assets, such as real estate properties or commodities, as digital tokens on a blockchain. Tokenization allows increased liquidity, fractional ownership, and easier transferability of these assets. It has the potential to democratize investment opportunities and make previously illiquid asset classes more accessible to a broader range of investors. [1]

Carbon credits are one such asset class that has seen increasing interest. They represent certified greenhouse gas emissions reductions resulting from carbon reduction projects in the physical world. These carbon credits can then be used by individuals or companies to offset their carbon emissions, thereby retiring the credit. Through this mechanism, carbon credits have become a widely used instrument in the fight against climate change.

The carbon market is a collection of actors and exchanges that facilitate the purchase and sale of carbon credits. It consists of two distinct markets: the compliance market

and the voluntary market (VCM). The compliance market originated as a tool to hold corporations and countries accountable for their emissions. It is governed by a regulating entity, which sets emission reduction targets that are legally binding and enforced. The voluntary market on the other hand enables individuals and companies in the private sector to buy carbon credits and thus compensate for (or neutralize) their emissions beyond mandatory levels. It also offers investors the opportunity to trade carbon credits and promote global decarbonization efforts by directing funds to new carbon reduction projects. As investments grow, it attracts new projects and leads to a flywheel effect. Therefore a well-established carbon market leads to accelerated development and implementation of sustainable solutions worldwide. [2] [3] The traditional carbon market has several shortcomings which inhibit its efficiency and scalability. One of the main issues is that the current market relies on over-the-counter brokers, who sell carbon credits on rate sheets, without utilizing exchanges or other market mechanisms. This means that each transaction requires individual purchase agreements, leading to a fragmented and inefficient market. [4] Another problem is the complexity and lack of standardization in the carbon market. There are multiple vintages, projects, and methodologies involved, which can lead to confusion and difficulties in verifying the legitimacy of carbon credits. This complexity also increases the risk of double-counting of credits, where sustainability claims are made multiple times against the same carbon credit, undermining the integrity of the market. [5]

The tokenization of carbon credits presents a potential solution to these shortcomings. Carbon certificates are brought on-chain via “carbon bridges” which are connected to traditional registries like "Verra" and "Gold Standard". [5] This creates an on-chain market where buyers and sellers can interact directly, eliminating the need for intermediaries as was needed in the traditional VCM setup. Instead, the market is accessible to anyone who can connect to the network using a wallet. This reduces transaction costs and time, as well as increases transparency, as all transactions are recorded on the blockchain and can be easily audited. This contributes to reducing fraud, as the process does not require a third party to be trusted. The open and public nature of distributed ledgers also serves as a crucial advantage against the double-counting of carbon credits. [5]

Furthermore, tokenization allows for greater liquidity and fungibility of carbon credits. Tokens can be easily traded and exchanged, enabling more efficient price discovery and market participation. [5] This liquidity attracts more investors and project developers, facilitating the flow of capital into carbon reduction projects. Overall, the tokenization of carbon credits addresses the inefficiencies and lack of transparency of the traditional VCM setup. It provides a more accessible, efficient, and liquid market for trading carbon credits, ultimately accelerating the transition to a low-carbon economy.

1.2 Outline of Contribution

The thesis presents a decentralized lending and borrowing protocol called "Green DeFi" that supports carbon tokens as collateral. By integrating carbon tokens in a DeFi product, the protocol can drive the adoption and usage of on-chain carbon tokens. Its smart contracts can provide a building block for further innovation with carbon credits, a concept known in blockchain development as composability.

1.3 Structure of the Thesis

Chapter 2 will give a general background of lending protocols, their functioning and components, as well as token economics. It will further explain the benefit of an integration with DeFi for the carbon market and explore other existing protocols and the approach they took. Chapter 3 examines the requirements of the protocol with a specific focus on its token (economics) design and governance system. Chapter 4 will explain the design of the protocol. It describes the architecture of the lending, tokenization, and governance smart contracts and their functioning and interaction. Chapter 5 describes the implementation of the protocol. It focuses on the implementation of the lending pool and vote escrow contracts. Chapter 6 evaluates the results of the work. Chapter 7 presents the conclusion of the thesis.

CHAPTER 2

Background

This chapter presents a technical background of lending protocols, introducing and explaining terms related to their functioning. Furthermore, it reviews different approaches to the design of token economics. Lastly, it will investigate the integration of carbon tokens with DeFi.

2.1 Lending protocols in DeFi

Lending protocols are a specific type of protocols that enable users to deposit, lend out assets, earn interest on them, and borrow assets against collateral. In traditional finance, lending and borrowing happen through financial institutions. Two aspects of lending and borrowing are interest payments and vetting borrowers for trustworthiness. [6] A blockchain protocol consists of multiple smart contracts deployed on the blockchain network and interacting with each other and external contracts. There is no restriction on who can call a contract, they are *permissionless*. Therefore anyone is able to call a lending protocol's contracts to get a loan. Another mechanism is needed to facilitate lending and borrowing, secure loans and ensure trustworthiness of debtors. This has insipired novel designs such as the Lending and Borrowing Protocol "Aave", that the "Green DeFi" protocol will be based on.

"Aave" employs a pool-based strategy, where both *deposits* and *borrows* are made to and from a single pool holding multiple asset reserves. Lenders receive special "AToken" that represent their carbon deposit and serve as their claim to the assets plus any accrued interests. Borrowing is possible once users have locked assets as *collateral*, where the amount deposited has to be greater than the amount (plus interest) borrowed, otherwise, a loan is considered *under-collateralized* and can be liquidated. Liquidation involves up to 50% of a borrower's debt to be repaid from their *collateral*.

2.1.1 Assets and Reserves

"Aave" allows users to deposit a cryptocurrency asset into the protocol's smart contracts and then use this deposit as *collateral* to borrow other assets against. Assets are deposited into a *lending pool* contract which holds each asset in its own *reserve*. *Reserves* are funded by lender's deposits and loan out funds to borrowers. This design is reminiscent of the liquidity pools in decentralized exchanges and does not require an order book where the two parties of a transaction are matched up by amounts. Instead, users can lend and earn interest, as well as borrow and repay loans plus interest at any time. [6]

2.1.2 Reserve Configurations

Each reserve has a *configuration* describing the conditions under which it can be used as collateral. As part of this configuration, the *Loan-to-Value ratio (LTV)* describes the maximum percentage of collateral value that can be used to borrow, i.e. the maximum amount of collateral that can be converted into a loan for a different asset. [7] For example, at 75% LTV, against a collateral of 1 ETH, a user can borrow 0.75 ETH. For a user with $TotalCollateral$ collateral and $TotalDebt$ loans in their account, the protocol requires:

$$LTV * TotalCollateral >= TotalDebt$$

Loans are *over-collateralized*, meaning that to take out a loan users have to supply collateral in amounts valued higher than the loan. A *liquidation threshold (LT)* is specified in the configuration, which gives the percentage of the collateral at which a loan would be considered *under-collateralized* for that asset. [7] As the market and prices for different assets are constantly evolving, once a loan has been taken out for a specific asset if the value of the collateral assets drops enough or the value of the loan plus interest rises enough, the loan can be liquidated. [8] To this end, a *health factor* is calculated for the position:

$$HealthFactor = TotalCollateral * \frac{LT}{TotalDebt}$$

A *health factor* of less than 1 indicates that the debt has crossed the *liquidation threshold* of the collateral and cannot be covered by it anymore, hence it can be liquidated to maintain solvency. [7]

2.1.3 Liquidations

During liquidations, the protocol allows third parties acting as *liquidators* to partially pay off a debtor's position. In return they can claim a part of the debtor's collateral at a discount: The reserve configuration defines a *liquidation penalty* which acts as a bonus for the *liquidators* - upon repaying the debt, the *liquidator* can claim the bonus and thereby get the collateral at a discount. [8] This incentive scheme saves the protocol the gas costs of liquidation transactions and the resources needed to monitor for liquidations

itself. [8] For example, if Alice deposits 10 ETH and borrows 5 ETH worth of another asset against it, if her health factor drops below 1, she gets liquidated. A *liquidator* pays off 50% of her debt, which amounts to 2.5 ETH worth of the asset. The ETH reserve defines a *liquidation penalty* of 5%, so the *liquidator* can claim a total of 2.5 ETH + 0.125 ETH of collateral. [8]

For liquidations to be profitable for *liquidators*, the position's debt cannot exceed its collateral value. *Liquidators* can repay debt on behalf of the user only after the liquidation threshold is crossed and only before the debt exceeds the collateral. [8] If this time window is too small and the loan goes insolvent before liquidation happens, the protocol will be left with a bad loan. This is generally undesirable for the protocol since the debt represents assets that people deposited into the pool but cannot withdraw anymore. [8] Therefore, choosing a proper liquidation threshold is critical for the protocol such that *liquidators* can be afforded enough time to repay debts on over-collateralized positions. [8]

The risk parameters around liquidation have to be determined individually for each asset. Changes in market conditions as well as governance proposals are a factor in this process. Assets vary in volatility. Among the least volatile ones are stablecoins, which are assets that are pegged in value to some external asset (for example, fiat currencies or commodities), followed by ETH. These two assets enjoy the highest LTV at 75% and a liquidation threshold of 80%. [7]

In DeFi lending, each position is opened against collateral assets and over-collateralized and there is no repayment schedule like in traditional finance. This creates new investment opportunities for users, such as putting leverage on their asset. If a user wants to *long* an asset, for example, MATIC (the "Polygon" network's native token), and is expecting its price to increase, they can deposit MATIC and take out a stablecoin loan against it, for example, the USD backed USDC token by "Circle", swap the USDC for MATIC and deposit it for another stablecoin loan. If the price of MATIC increases, the user obtains USDC for cheaper (USDC has a stable price as a stablecoin), pays back their loans, and reclaims the higher value MATIC for a profit. Similarly, if the user wants to *short* MATIC, they can deposit USDC to borrow MATIC, swap it for USDC, and deposit that as collateral for another MATIC loan. If the price of MATIC now falls, their loans are cheaper to repay and they reclaim the USDC deposits (which have a stable price) at a profit. Whether *shorting* or *longing* an asset, users are able to increase their position by using their loaned assets as leverage. This is possible as long as the health factor of the position is large enough. [6]

2.1.4 Interest Rates

Interest rates are one of the key components of lending protocols and represent an important incentive mechanism to attract users and liquidity. A contract called "AToken" manages interest rates when a user lends funds to the pool. New "AToken" are minted when a user deposits assets into the lending pool and represent the user's share of the

total funds in the reserve. The tokens are burned when a user withdraws their funds.

For interest accumulation, each reserve contains a *LiquidityIndex* which increases over time as interest accrues in the pool. It represents the principal assets and any cumulative yield generated by the pool since its inception. At each user action (deposit, borrow, repay, withdraw), the *LiquidityIndex* is recalculated by cumulating interest since the last user action using the current interest rate (*LiquidityRate*)[6]:

$$LiquidityIndex = \left(\frac{LiquidityRate * \Delta time}{31536000} + 1\right) * LiquidityIndex_{prev}$$

When a user wants to deposit new assets of amount A into the pool, the "AToken" contract will take ownership of the assets and mint the user new tokens. Their amount is determined by the following formula:

$$amount = \frac{A}{LiquidityIndex}$$

The process mints the user tokens that are proportional in value to the current total interest. The amount is a scaled representation of the user's assets and retrievable as *scaledBalanceOf*. These tokens then entitle the user to their original assets and the interest accumulated on them. When a user wants to withdraw their assets, the *LiquidityIndex* is calculated with the current timestamp to get a representation of the total interest in the reserve. Then their *scaledBalanceOf* is scaled back with the *LiquidityIndex* to get the amount of underlying assets they are owed by the protocol:

$$amount = scaledBalanceOf * LiquidityIndex$$

As an example, consider the case where a user deposits 100 USDC. At the time of the deposit $LiquidityIndex = 1$, so the user receives

$$scaledBalanceOf = \frac{amount}{LiquidityIndex} = \frac{100}{1} = 100 AToken.$$

When the user wants to withdraw their assets after some time has passed $LiquidityIndex = 1.1$, every unit has produced 0.1 units of income. Therefore the user can receive

$$amount = scaledBalanceOf * LiquidityIndex = scaledBalanceOf * 1.1 = 110 USDC$$

gaining 10 USDC of interest.

The interest rate (*LiquidityRate*) of the market is updated to reflect changes in the overall reserve's liquidity, such as when withdrawing assets, borrowing, depositing, or repaying. As the *LiquidityIndex* is recalculated based on the current *LiquidityRate*, this ensures that a user's balance is adjusted to reflect changes in the overall reserve's liquidity and its interest rates. [6]

2.2 Tokenomics

Token economics (*Tokenomics*) refers to the design of the native token issued by a protocol, the incentive systems that determine how the token will be distributed, and the utility of the tokens that influence its demand. [9] It is implemented through the protocol's smart contracts. *Tokenomics* plays a vital role in incentivizing user participation (for example, liquidity provision), rewarding value contributions, and coordinating and aligning different stakeholders of the protocol.

2.2.1 Token Emissions

A key part of the token design are general parameters around the token economy. *Token emission* in a blockchain ecosystem refers to the creation of new tokens and release to the public within a specific timeframe. The *emission rate* directly impacts the supply and demand for the cryptocurrency. *Fixed supply, infinite*, and *deflationary token emissions* are different approaches to managing the maximum supply of tokens in a blockchain ecosystem. [10] The total supply of a cryptocurrency is the number of tokens in circulation and available for trades added to the number of tokens that are minted (i.e. existing and not burned) but not yet distributed. [11]
Maximum supply refers to the maximum amount of tokens that will ever exist in the lifetime of the cryptocurrency. Once the maximum supply is exhausted, no new tokens will be produced or mined. It is usually defined in the genesis block. [10] *Fixed (maximum) supply* is the most common type and is implemented in most cryptocurrencies. Once the total supply reaches an upper predetermined limit, it can never be increased or decreased, regardless of demand. Bitcoin for example has a fixed supply of 21 million coins. [10] The current total supply is about 80% of this limit and is technically still inflating. However, the purchasing power of one bitcoin is increasing over time as the demand increases and supply is limited. [12]
Infinite supply cryptocurrencies have no maximum supply, the number of their tokens can increase indefinitely. Their supply is inherently inflationary. It is often seen in utility tokens, where the supply is not limited by the number of coins but by the number of users. [10] Ether is an example of an infinite supply token, although its issuance is fixed at roughly 1,700 ether/day after the merge [13], meaning that its inflation rate is decreasing with time as the total supply increases. Additionally, EIP-1559 ("Ethereum Improvement Proposals") has altered the economic nature of the Ethereum token by incorporating the burning of a fraction of the gas fees per transaction, such that the burn rate effectively outweighs the inflation rate and renders the token deflationary, as more Ether is burned in maintaining the network activity than the amount of Ether is entering circulation. [14] Such a *deflationary supply* which decreases over time, is designed to create scarcity and price appreciation.

2.2.2 Liquidity Mining

Liquidity mining offers native token rewards to users (*liquidity providers*) for providing the protocol with deposited assets that the protocol can use to facilitate their decentralized finance services. Most protocols allocate a large proportion of their native tokens into *liquidity mining* incentives to bootstrap themselves and attract more users. [15] This token allows the users to participate in the protocol's governance (*governance token*) as the main utility of the token and where its value is derived from. This makes them very inflationary. [16]
Apart from a token reward, incentives for users can also take the form of profits gained from receiving a percentage of the transaction fee and interest from lenders. The practice of staking or lending cryptocurrency assets to protocols to generate high returns or rewards is called yield farming. [17] These returns can be expressed as *annual percentage yield (APY)* or *annual percentage reward (APR)*. *Liquidity mining* therefore occurs when participants earn additional compensation in the form of native token rewards. [17] It is popular on decentralized exchanges, which offer pools of two different assets each that enable one to swap one asset for the other without having to match up buyer and seller. To provide liquidity to these pools, users deposit funds into them and receive *liquidity provider (LP) tokens*, which represent their deposit in the pool and on which they can farm protocol rewards. [17]

Disadvantages

A disadvantage of liquidity mining is that it requires the protocol to allocate token emissions to offer rewards to liquidity providers and attract them. They have to be sustained by a high token emissions rate, which is expensive and unsustainable in the long-term for the protocol. It also creates a "race to the bottom", as the protocols compete against each other to offer the highest incentives, further diluting their token supply and decreasing the value of their token. This has been termed the "mercenary capital problem". [18] Yield farming users are seeking to maximize their returns or rewards. This leads them to 'dump' the token as soon as the rewards dry up, diluting the market supply and causing sell pressure on the token and price volatility. This is further undesirable for the health of a protocol. [15] Users also need to sell the tokens to realize the rewards. [19] Furthermore, users need to consider the impermanent loss incurred by providing liquidity. Impermanent loss is the opportunity cost of holding onto an asset for speculative purposes versus providing it as liquidity to earn yield. With volatile cryptocurrencies, it is almost impossible to avoid impermanent loss. For an asset that increases or decreases in value after being deposited by a user, the user is at risk of not profiting from this gain or even losing money. For example, Ethereum can double its value within five days but the yield from providing it as liquidity may not cover this gain. [20] As many newly launched protocol tokens are highly volatile, there is an increased risk of impermanent loss for third-party liquidity providers which represents a misaligned incentive structure for these users, who don't have many options to manage the risk of providing liquidity

and yield farming. This makes any liquidity mining incentives even more critical to attract liquidity providers. [21]

A contract that implements yield farming and that can be used to illustrate mercenary liquidity is “MasterChef.sol”. It was first deployed by the decentralized exchange "SushiSwap" to draw liquidity from its competitor "Uniswap" in the 2020 “DeFi Summer” when yield farming became popular and has since been forked and used by many other projects. A 2021 aggregate analysis (shown in Figure 2.1) of all "Masterchef.sol" contracts on-chain by Blockchain Analytics platform “Nansen” has quantified how fleeting the capital deposited into so-called “farms” really is. According to their findings, the majority of yield farmers appear to exit within 5 days of entering a farm. Of the people who enter a farm on the day it launches, 42% exit it within 24 hours. By the third day, 70% of these users would have withdrawn their funds from the contract. This highlights how short-lived the liquidity achieved through yield farming is. It also illustrates how funds are progressively withdrawn as the rewards are decreased as more users join the farm. [22]

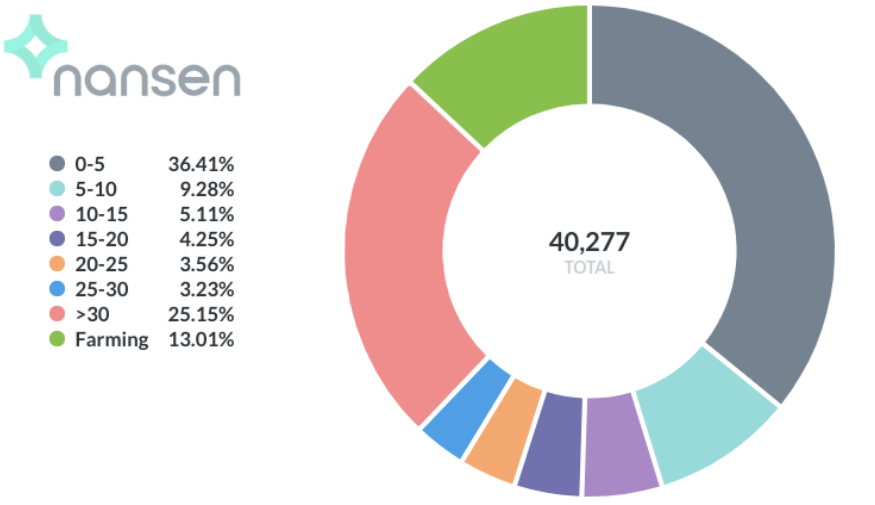

Figure 2.1.: Showing the distribution of the number of days that capital entering "Masterchef.sol" contract or forks remained in the contract [23]

Despite the tradeoffs, liquidity mining remains one of the most popular ways for projects to incentivize liquidity because of their tried and tested implementation, familiar mechanism, and ease for users. [19]

2.2.3 DeFi 2.0 and Protocol-Owned Liquidity

Protocol-owned liquidity is a new approach to attaining liquidity for protocols where protocols acquire their liquidity from the market or rent it from other protocols instead of diluting their token supply in exchange for mercenary liquidity. Token emissions get

aligned with actions that are beneficial for the protocol. This is also referred to as “Defi 2.0”. [15]

Bonding

Users who provide liquidity receive Liquidity Provider (LP) tokens, these represent their assets and entitle them to them. *Bonding* involves users selling their LP tokens to the protocol to receive the protocol's token at a discount. For example, the protocol sells 3.5$ worth of native tokens for 2.5$ worth of LP tokens. The $1 discount provides an incentive for the user to bond the native token. As the protocol is now in possession of the LP tokens, it has gained ownership of the liquidity, hence achieving protocol-owned liquidity. This not only removes the possibility for liquidity exits but also allows the protocol to build a pool of fee-generating liquidity, as it is not required to pay a percentage of the transaction fee (in the case of the "Green DeFi" lending protocol, this would be interest paid on the loan) to the depositors. Furthermore, the protocol gains control over two levers: the rate at which tokens are exchanged for liquidity and the total amount of liquidity exchanged. If the token price is falling due to increasing supply and many users *bonding* and acquiring discounted tokens, the protocol can change the discount rate (even reversing it into the negative). Similarly putting hard caps on the amount of bonds can provide more control over the token expansion.

Bonding was first pioneered by "Olympus DAO" which build a treasury with protocol-owned liquidity to back their decentralized reserve currency OHM. [24] Since then, several projects have used *bonding* to achieve liquidity for their use case. [19] In preventing arbitrage opportunities for users who bond, these projects usually implement a vesting scheme for the token. A vesting scheme is a predetermined timeline or condition under which the tokens become fully transferable or tradable. By implementing a vesting scheme, the project ensures that users who bond the tokens cannot immediately sell them for a profit, thereby discouraging arbitrage and promoting long-term commitment to the project. [25]

Vote-Escrow Tokenomics

"Curve Finance" offers a decentralized exchange focused on stablecoin trading with low fees and slippage. [26] Due to the nature of the *Automated Market Maker* trading they offer, this requires them to have deep liquidity pools. Their system to incentivize users incorporates rewards for liquidity providers but goes beyond mere liquidity mining. Their reward token can be (vote-)locked in a (vote-)escrow contract, in exchange for the user receiving voting power in the form of a *vote-escrow* token, veCRV. In combination with the staking of the LP tokens, this unlocks additional utility for the CRV, as this voting power can be used both to boost as well as vote on which pools receive liquidity mining incentives. Users can earn a boost of up to 2.5 on the rewards they receive as a liquidity provider. The protocol also distributes 50% of the admin fees it earns from trading to

users based on their locked (staked) CRV. Therefore the utility of CRV is threefold as shown in Figure 2.2 - boosting, voting, and earning fees through staking.

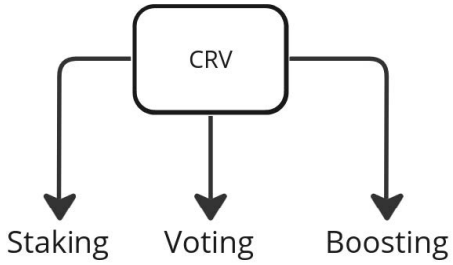

Figure 2.2.: Utility of CRV token [27]

As the only way to obtain veCRV is by vote-locking CRV, and veCRV is designed to be non-transferrable, this increased the demand for CRV tokens. In the past, this led to a situation called the "Curve Wars" where different protocols bribe CRV holders to lock and vote with their voting power to distribute the token emissions to the liquidity pool of the specific protocol's token. As such, vote-escrow tokenomics represents a way to incentivize liquidity providers by *other protocol's tokens* being spent.

2.3 Integrating Carbon Tokens with DeFi

A benefit of having carbon credits on-chain is that they can be supported by any protocol on the blockchain network. This circulates the token and contributes to its adoption, hence creating a market for it. For instance, carbon credits could be used as a collateral asset for a lending and borrowing protocol. This drives up demand, increasing the price of carbon tokens and thereby making it more expensive to emit greenhouse gases. Such an integration with a DeFi protocol establishes carbon tokens as a new asset class that provides new opportunities for investors to participate in sustainable finance projects.

2.3.1 Similar Protocols

"KlimaDAO" has built a reserve currency called KLIMA backed by carbon assets. The idea of KLIMA is to make the price of carbon and emitting carbon expensive by becoming a purchaser of carbon tokens. KLIMA can be minted at a discount by depositing carbon tokens into the KlimaDAO treasury, in a process called bonding. This means that each KLIMA is backed 1:1 with a specific amount of carbon emissions ("Intrinsic Value" or IV). After bonding users can stake their KLIMA with the protocol and earn rewards, which incentivizes them to not sell the KLIMA as it was generated at a discount (and also could be trading above the market value of a carbon tonne/IV). The surplus worth

(seigniorage) that the protocol retains in this way is distributed to the stakers.
The protocol serves as the "decentralized central bank" of the token, with the ability to expand and contract supply. In situations where KLIMA is trading above the IV, the protocol will expand supply and sell KLIMA to the market, for the reverse situation, it buys KLIMA under the IV by selling its carbon assets. [28][29]

2.3.2 Differentiation

The goal of "KlimaDAO" is to increase the price of carbon. They built a financial instrument - their KLIMA reserve currency - for this purpose. It is a fork of "OlympusDAO", which builds a decentralized reserve currency that is deeply liquid and fully backed by stable assets and is a project designed to make it easy to create such asset-backed stablecoins, as with the case of KLIMA. [29] [24]
The "Green DeFi" protocol will differentiate itself from "KlimaDAO" by focusing on building a protocol that offers a service in DeFi - lending and borrowing - instead of a stablecoin token product being at the core. Lending and borrowing are facilitated through a pool of assets which can include carbon tokens. Therefore it only provides an integration of carbon tokens and does not seek to build a new financial instrument to be traded at the highest price. As such, the price of carbon tokens is left to be determined by the market. The tokens of the "Green DeFi" protocol will be described in the next chapter.

CHAPTER 3

Analysis

The goal of this chapter is to describe the requirements of the "Green DeFi" protocol. It specifically focuses on the requirements of the tokenomics system, including the token and voting escrow contract requirements.

3.1 Requirements of the Protocol Token

The "Green DeFi" protocol needs to introduce its own token to facilitate governance, trading, and market making. The token's utility and distribution must be well-defined and transparent. Users should have a clear understanding of how token emissions, rewards, and governance functions work within the ecosystem.

The token should follow the standard "ERC20" specification for fungible tokens. This gives it all the necessary functionalities to be transferred between users, so that it can be traded in the open market, as well as be minted and burned to control its token supply. The token will also be needed to vote on governance proposals for the protocol. Most importantly the token is an instrument and asset of value to incentivize users to participate in market making, i.e. creating liquidity in the protocol's pools and reserves. The "Green DeFi" protocol needs to hold and attract the necessary funds from liquidity providers before users can use the platform to the full extent. It needs to ensure that trades are possible without large movements in price due to supply and demand. [19] The protocol token and the crypto-economic incentives built around it play a key role in this.

The "Green DeFi" protocol can issue its own token ("GREEN") which will have a limited maximum supply. The protocol needs to carefully manage token emissions to ensure a balance between incentivizing users and maintaining the token's value. A fraction of the maximum supply of tokens can be created and sold in an *Initial Coin Offering (ICO)* similar to an IPO to fund the project.

GREEN could be used directly in a governance function and to provide rewards to users in a liquidity mining scheme, but there are multiple reasons why this approach is not ideal. High token emissions rates, such as those needed for yield farming, are costly and unsustainable to the protocol. Token inflation erodes value and discourages users from participating and should therefore be avoided. Instead, incentives should be designed to encourage long-term commitment by users.

3.2 Requirements of the Voting System

To avoid the problems associated with traditional liquidity mining methods, it would be desirable to acquire liquidity from the market without diluting the "Green DeFi" protocol's own token supply too much. This can happen through protocol-owned liquidity.

The "Green DeFi" protocol can aim to create an incentive system similar to the "Curve Wars". GREEN holders can vote on which asset to allocate rewards to. A protocol that wants to ensure liquidity of its native token on the "Green DeFi" platform can bribe the GREEN token holders to vote for the asset that is its token. Liquidity providers of this asset receive GREEN rewards as an incentive, similar to in the "Curve wars".

Voting Contract

The voting contract can record users' votes and be queried by the other contracts so that the protocol can determine which assets users favor the most. The requirements of the voting contract that registers user's votes are as follows:

- i) Users cannot vote twice with the same GREEN token.

This prevents the abuse of the voting system. Because GREEN is a fungible token, there is no method to identify individual tokens or track them when they are transferred between users. Hence, there can be no guarantee of i) when using GREEN directly to vote on reserves. Staking GREEN presents an alternative that excludes the possibility of "double-counting" GREEN. Staking involves users locking assets into a pool or contract for a specific period of time to earn rewards, or in this case voting power. Staking also takes GREEN out of circulation to manage inflation. This reduces the overall token supply and maintains its value. The third benefit of staking is that it presents a mechanism that discourages short-term thinking in users, as they have their tokens locked in the protocol. The more tokens, the more voting power the user acquires. Therefore the requirements are:

- ii) Users acquire voting power through staking GREEN.
- iii) Voting power must depend on the amount of GREEN staked.

Voting power is the possibility to vote, like having a certain number of votes that can

count towards proposals (in this case proposals are reserves that are voted on). Achieving voting power and voting on proposals are separated to encapsulate different operations and user interactions.

- iv) Users realize their full voting power as soon as they stake GREEN.

It would be possible to design a voting system where users have to wait for a certain duration after staking to "claim" their voting power. But this is inconvenient for the user, as they are not able to immediately use their voting power and would have to schedule it for later. This creates "friction" in the user experience and discourages users from participating in the governance. This situation differs from traditional token vesting schemes, because voting power should not be transferrable like a token, and can therefore not be traded or sold on the market for profit. As contracts are permissionless, there would be no restrictions in place to prevent a user from buying and holding onto an excessive amount of this token, for example for speculative purposes. A situation like this is undesirable for a voting system. Voting power should not directly be used as a financial investment by users, instead, it should reflect the individual's capacity to vote.

- v) Voting power is bound to the stake of a user and cannot be transferred between users.

What is the relationship between time and voting power? If voting power is independent of time and will never change, it is a similar situation as above, where the voting power grows over time. In this situation, users lack an incentive to participate immediately in the governance. There is no difference in whether users vote now or later. Therefore, voting power should decrease with time, the decrease should be a factor of the amount of tokens A in the stake. So, of the function $W(t)$ that defines the voting power over time, it is known that $W'(t) = -A * b$, therefore $W(t)$ must be linear.

- vi) Voting power is a linear function of time.

$W(t)$ has its maximum at $t = 0$. Therefore, the voting power a user gains are highest at the moment of staking and afterward decreases linearly.

veGREEN token

The voting contract acts like an escrow contract for the GREEN token ("voting escrow"). It holds onto the assets while the user realizes their voting power. When the stake expires, the assets are released. In the meantime, they can be represented by the voting contract as a new token, as tokens are one of the core primitives of blockchain. Voting power can therefore be represented as a non-transferable token "veGREEN" that is minted when a user stakes GREEN into the "voting escrow" contract.

Define:

- vii) 1 veGREEN is equal to 1 GREEN locked for the maximum duration of $t_{max} = 4$ years (at the moment of staking).

It follows that 250 veGREEN $\widehat{=}$ 1000 GREEN locked for 1 year, or 250 veGREEN $\widehat{=}$ 250 GREEN locked for 4 year. Importantly, for a GREEN stake of amount A and duration D, it holds that

$$W(0) = \frac{A * D}{t_{max}} \tag{3.1}$$

Using integration, *vi*) leads to

$$W(t) = -A * b * t + C \tag{3.2}$$

with constant C. Combining 3.1 and 3.2 leads to

$$\frac{A * D}{t_{max}} = W(0) = -1 * b * 0 + C = C$$

Voting power should expire at $t = min\{D, t_{max}\}$. Plugging in for $A = 1, D = 4$ leads to

$$0 = W(t_{max}) = -1 * b * t_{max} + 1 => b = \frac{1}{t_{max}}$$

Therefore for voting power function

- viii) Voting power gained by staking GREEN with amount A and duration D is given by:

$$W(t) = -A * \frac{t}{t_{max}} + \frac{A * D}{t_{max}} \tag{3.3}$$

for $0 <= t <= t_{max}$. The more GREEN users stake and the longer they stake for (under the maximum duration), the more voting power users gain.

Other Contract Behavior

After the expiry of the voting power, the voting escrow contract should allow users to withdraw their GREEN tokens. The stake has to expire before withdrawal from the contract is possible, a stake that was created may not be "nullified" to withdraw the tokens earlier. Otherwise, an attacker could stake, vote, and immediately withdraw their tokens without their vote becoming invalid in the governance systems. The staked GREEN should be withdrawn before the user can create a new stake. To save the user the gas cost of these transactions, users should be given the ability to extend existing, non-expired staking positions either by amount or duration. This also allows to offset the decay of voting power or increase it if a user deems it insufficient. Ultimately, users who lock more tokens and who lock them for longer are rewarded for their long-term commitment. This aligns GREEN token emissions with actions that are beneficial to the protocol.

3.3 Tokenomics System

In the voting escrow contract users lock their GREEN tokens in exchange for voting power (veGREEN). This voting power can be used to vote on the allocation of GREEN rewards on different assets in the pool. An additional incentive is provided when introducing reward boosting using veGREEN.

Voting on different assets requires a mechanism to track the number of votes for each asset. This is facilitated by the introduction of a *gauge weights* associated with each asset. The *gauge weight* can be used to determine how many votes an asset was allocated by users. The protocol can then distribute the GREEN rewards. An overview of the whole tokenomics system is presented in Figure 3.1. After depositing the carbon assets, the liquidity provider receives interest from lending and the GREEN reward from the pool. Staking this GREEN gives them the voting power to vote on different gauge weights. This increases their rewards by boosting and voting on *gauge weights*, this feedback loop is shown in red. This system presents an innovative incentive structure to encourage engagement by users and acquire sustainable liquidity.

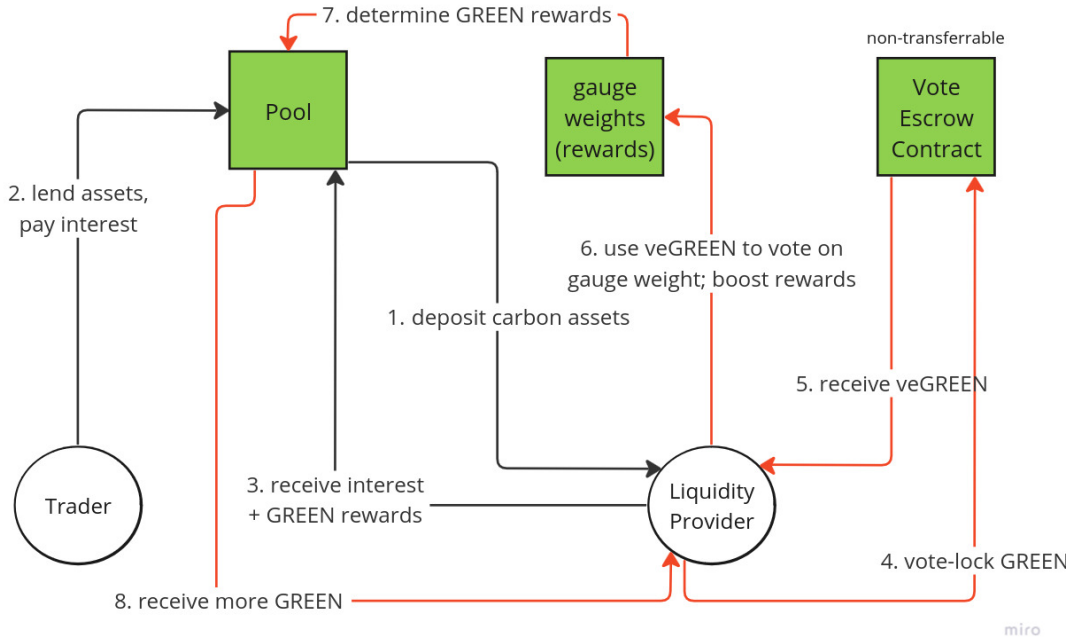

Figure 3.1.: Vote Escrow Tokenomics Incentive Diagram for the "Green DeFi" protocol [27]

CHAPTER 4

Design

This chapter describes how the protocol is designed and the motivations behind particular design choices. It covers the main lending pool contracts in the first part and the governance and tokenomics contracts in the second part. The "connective component" between both is the `AToken` contract of the lending pool that represents the token to be locked in the voting escrow contract.

4.1 Lending Pool

This section will describe how the borrowing and lending mechanism of the protocol is designed, including the pool and associated token contracts.

4.1.1 Pool and Configuration

The contracts that manage the lending and borrowing mechanism of the protocol can be divided into a main pool contract which users interact with (including all the necessary storage constructs) and a configuration contract which allows the protocol's admins and deployers to configure and initialize the lending pool and its reserves. The protocol first deploys a `LendingPoolAddressesProviderRegistry` contract that acts as a registry for multiple `LendingPoolAddressesProvider` contracts each representing its own market. Within a provider the owner (admin) can set a `LendingPoolConfigurator` and `LendingPool` contract, using a *proxy pattern* to preserve the upgradability of the contracts. The `LendingPool` is the contract with user-facing methods, such that users and third-party contracts can interact with it on-chain. `LendingPool` can be part of a series of smart contract calls that together provide a new DeFi service built on the carbon-backed lending protocol. *Composability* refers to the ability of different decentralized applications (*dApps*) and protocols to seamlessly interact and integrate. `LendingPool` provides methods to deposit, borrow, repay, and withdraw assets.

4.1.2 Proxy Design

As smart contract code is immutable on-chain, the code cannot be rewritten once a contract is deployed. It is however possible to restructure the code into different contracts such that the logic can be upgraded while preserving the old storage. This is useful e.g. for managing bugs and smart contract vulnerabilities by providing the ability to upgrade the program logic while preserving its user-facing address on-chain. With the *proxy pattern*, the program is split into two contracts, an upgradable *implementation* or logic contract, and an immutable storage contract or *proxy* (contract). The *proxy* contract is a minimal contract containing the address of the *implementation contract* in its storage, it delegates all its calls to the address.
The Contract *Application Binary Interface* (*ABI*) is the standard way to interact with contracts in the Ethereum ecosystem, both from outside the blockchain and for contract-to-contract interaction. Data is encoded according to its type, as described in this specification. [30] Smart contract interactions are achieved by populating the *calldata* with information about the function signature and arguments according to this specification. To call the *proxy*, the *implementation* contract's *ABI* serves as the specification. A transaction to the *proxy* with the resulting *calldata* triggers the execution of the *fallback* function, which is the function the *Ethereum Virtual Machine (EVM)* executes in the contract if the decoded function signature of the *calldata* matches none of the contract's functions. Therefore the *proxy* contract must only implement this *fallback* function. It also needs a setter to modify the implementation address state variable.
Inside the *fallback* function, it forwards the call to the *implementation* contract using *delegatecall*, a Solidity instruction that executes the implementation logic in the environment (including storage) of the original call to the *proxy* contract. Therefore the *proxy* contract provides nothing else than the state of the program through changes to the storage that are achieved using the logic of the *implementation* contract, it does not hold any own logic, except the aforementioned one to delegate to and set the implementation address. This way upgradability is achieved for all the contracts that manage the critical components of the protocol such as the `LendingPool` and `LendingPoolConfigurator` but also the token contracts described in the next chapter.

4.1.3 Tokenization

Tokens are one of the main primitives of blockchain. Tokens can be used in DeFi to represent shares, debt, or interest. This section investigates the design of one of the protocol's main lending tokens.

Fungible Tokens and ERC-20

Tokens in DeFi should be fungible and act like a currency, such that each token is interchangeable instead of representing an individually valued asset on the market. Formally, fungible tokens are represented by the `ERC-20` token standard, which describes the be-

havior of a token contract where each token is the same in value and type as any other. These are needed to make the tokens of the Green DeFi protocol a currency that can be traded on the open market.
These are the specific methods of the `ERC-20` standard:

- `function name() public view returns (string)`
- `function symbol() public view returns (string)`
- `function decimals() public view returns (uint8)`
- `function totalSupply() public view returns (uint256)`
- `function balanceOf(address _owner) public view returns (uint256 balance)`
- `function transfer(address _to, uint256 _value) public returns (bool success)`
- `function transferFrom(address _from, address _to, uint256 _value) public returns (bool success)`
- `function approve(address _spender, uint256 _value) public returns (bool success)`
- `function allowance(address _owner, address _spender) public view returns (uint256 remaining)`

`ERC-20` Interface [31]:

- `name()` and `symbol()` give the token name and the shorthand symbol used to denote the currency, e.g. the symbol for Ether is `ETH`.
- `decimals()` define the smallest unit of the asset that denotes how divisible the token is. The standard decimal number of `18` (used in the protocol) means that `1` token is divisible into `10**18` units called `Wei`.
- `totalSupply()` can be used to retrieve the total balance of the token, i.e. the total number of tokens in existence in the network, including tokens that are locked and not in free circulation.
- `balanceOf()` can be used to retrieve the token balance of the provided address, as it is stored within the contract. Changes in the value of `balanceOf` reflect account movements of a user
- `transfer()` allows to transfer tokens from one account to another.
- `approve(), allowance(), transferFrom()` are used in the "approve and spend" workflow which requires a user to approve the transfer of assets first before a smart contract can transfer them out of their account.

The `AToken` design can serve as an example of how the protocol specifically follows the `ERC-20` specification.

AToken (Share Tokens)

The `AToken` contract is fully `ERC-20` compliant. Each reserve of a pool has an associated `AToken` deployed in the network and tokens are issued to users to represent deposited assets. Importantly, the `AToken` contract also manages the interest generation for this reserve: When a user deposits assets, the token contract stores a scaled version of the amount, representing their share of the reserve. The contract also holds a singular value representing the total interest on this reserve which is designed to be applied to the scaled representation to calculate the user's original principal plus the interest generated on these assets. The `balanceOf()` function returns this value for the provided address. Similarly, the `totalSupply()` returns the total assets including all accrued interest. The `transfer` function effectively transfers a user's shares of a reserve (their `AToken`) to another user, such that the recipient user owns the deposits associated with it and can withdraw the assets. This allows shares to be traded on the market.

4.2 Design of the Governance System

This section presents the design of the governance and tokenomics contracts of the protocol. The formal requirements, especially of the voting escrow contract, have been gathered in Chapter 3.

4.2.1 Overview

The governance contracts manage the allocation, distribution, and minting of the governance token of the "Green DeFi" protocol. They implement the incentive system presented in Fig. 3.1 and comprise the following:

1. `VotingEscrow:` Into this contract, users can stake (vote-lock) their GREEN and receive veGREEN (vote-escrow GREEN), which represent voting power within the contract. As described in Chapter 3, the vote escrow system calculates the voting power based on both lock duration and amount. Users can extend the lock by duration and token amount.
2. `GREEN`: The protocol's native token, distributed as a reward to liquidity providers.
3. `LendingPool:` This is the contract that holds the deposited assets (liquidity pools). Each asset has its own reserve.
4. `LiquidityGauge:` Each reserve has an associated `LiquidityGauge`. Users can allocate votes to individual gauges and deposit their LP tokens (`AToken`) into them. This unlocks the boosting and receiving of additional GREEN allocations from voting on these gauges. The `LiquidityGauge` therefore represents a fundamental component of the protocol-owned liquidity system as the holders of the liquidity.
5. `GaugeController`: This contract keeps track of all the `LiquidityGauge` and maps each gauge to an internal gauge weight. When users vote on a `LiquidityGauge` and

allocate it veGREEN, it is reflected in the gauge weight. The `GaugeController` then calculates the GREEN rewards for each reserve.

6. `Minter`: This contract is responsible for minting new GREEN rewards. The amount of new tokens is determined by the `LiquidityGauge` contracts, which tell the `Minter` how many LP tokens a user locked in the gauge. In the protocol-owned liquidity system, the user's rewards are determined by how much liquidity they lock.

4.2.2 Voting Escrow Contract and Token Design

The veGREEN token is designed as a non-transferable token used to represent the voting power of holders. Therefore it does not fulfill the `ERC-20` specification given earlier, as functions related to allowance and transfer of tokens are not given. Instead, the contract has the `name()` and `symbol()` of the veGREEN token, and a call `balanceOf()` returns a value for the user's voting power. This replaces the standard internal `_balances` mapping in an `ERC-20` token contract, which would be read out to obtain the number of tokens held by the user. This `_balances` is usually changed with a minting and burning operation within the contract, in the `VotingEscrow` contract this is replaced by a `createLock()` function, which creates a locked position for the user with some voting power/veGREEN balance. The voting power can then be queried using `balanceOf()`.

As explained in Chapter 3, voting power decreases linearly from the moment of locking. It is possible to read out the future voting power for a user from the contract with `balanceOfAt()`, this will return the calculated voting power at the specified time. A user can change their position by increasing the amount with `increaseAmount()` or duration with `increaseTime()`, hence preserving or increasing their voting power. If a user's position changes, the time and new voting power are recorded and can be retrieved with `getLastUserPoint()`.
After the position expires, a user can withdraw their GREEN using `withdraw()`. The current version of the contract also contains `addToAllowlist()` and `removeFromAllowlist()` functions to whitelist addresses (wallets) that are allowed to deposit to further prevent the token from being traded.

The full interface is as follows (events, enums, and `checkpoint()` have been omitted here and will be explained in Chapter 5):

```
1     interface IVotingEscrow {
2         enum LockAction {
3             CREATE_LOCK,
4             INCREASE_LOCK_AMOUNT,
5             INCREASE_LOCK_TIME
6         }
7         event Deposit(
```

```
8               address indexed provider,
9               uint256 value,
10              uint256 unlockTime,
11              LockAction indexed action,
12              uint256 ts
13          );
14          event Withdraw(address indexed provider, uint value
                , uint ts);
15
16          function addToAllowlist(address addr) external;
17          function removeFromAllowlist(address addr) external
                ;
18          function getLastUserPoint(address addr) external
                returns (int128 bias, int128 slope, uint256 ts);
19          function createLock(uint256 value, uint256
                unlockTime) external;
20          function increaseAmount(uint amount) external;
21          function increaseTime(uint newTime) external;
22          function checkpoint() external;
23          function balanceOfAt(address addr, uint ts)
                external returns (uint);
24          function balanceOf(address addr) external returns (
                uint);
25          function withdraw() external;
26      }
```

Source code 4.1: Solidity Interface of the VotingEscrow Contract from `contracts/protocol/governance/IVotingEscrow.sol`

CHAPTER 5

Implementation

The smart contracts of the "Green DeFi" protocol were implemented in `Solidity`, a statically-typed programming language that compiles down to `EVM` bytecode, and with the help of the application development toolkit `Foundry`. The contracts were unit-tested with the help of its testing framework `forge`. The test setup was a local fork in `Foundry` of the `Polygon` network `mainnet`, where the carbon assets are available. This allows the implementation and testing of the borrowing and lending of carbon assets as a proof of concept of the carbon-asset backed lending protocol. The source code for the protocol can be found under Appendix A.

5.1 Assets

The carbon assets used in the proof of concept were:

- `Nature Carbon Tonne (NCT)` by `Toucan Protocol`, available *here* on `mainnet`
- `Moss Carbon Credit (MCO2)` by `Moss.earth`, available *here* on `mainnet`

As price feeds are not available for both assets, they had to be provided using a mock Oracle contract, implemented in `contracts/mocks/oracle/PriceOracle.sol`. This fixed the price to the value on 8.8.2023.

5.2 Lending and Borrowing

Lending and borrowing were implemented using the core smart contracts of the "Aave V2" protocol. These contracts work together in the following way to implement the borrowing and lending mechanism:

1. The `LendingPool` provides the user functions for depositing assets into reserves, withdrawing, borrowing, and repaying loans. It uses a mapping of custom data types (Solidity structs) from the `DataType` library to hold the information for the

different reserves. This custom `ReserveData` type encapsulates all the information about the asset, such as the addresses of the associated token contracts (e.g. `AToken`, or `DebtToken` that represent debt) or the various metrics needed for the calculations around the protocol, such as `liquidityIndex` or `currentLiquidityRate`. Each `ReserveData` includes a `ReserveConfiguration` that specifies rules for the reserve, such as whether the reserve is active or frozen, whether borrowing is enabled, and in which mode (stable or variable interest rates). The configuration of the reserve for specific users, such as whether they allow an asset as collateral, is stored in the `UserConfigurationMap`.

The `LendingPool` executes the logic of the contracts found under `contracts/protocol/libraries/logic`. These contracts are called from the `LendingPool` while passing the configuration and reserve data:

2. The `ValidationLogic` library contract provides functions to validate different user actions. They ensure the actions comply with the configuration of the reserves. They also assert the financial health of the user's position.
3. The `ValidationLogic` contract calls the `GenericLogic` contract, which is a library that provides functions to calculate and validate the state of an individual user. The main calculation happens in `calculateUserAccountData`. It comprises aggregating the user's positions across reserves, counting their total collateral and debt in `ETH` value, applying factors such as the `liquidationThreshold` and `LTV`, and computing a `healthFactor` from the result of this process. This value is an indicator of whether a user's loans are at risk of defaulting.
4. The `ReserveLogic` contract is a library that provides functions for managing the state of each reserve. It includes functions to update the `liquidityIndex` based on the current state of the reserve and the passage of time, representing interest accumulation. It also calculates the updated `currentLiquidityRate` and `borrowRate` (stable and variable) for a reserve based on any changes that occurred in liquidity or debt.

These contracts are designed to work together to provide a comprehensive and secure lending and borrowing platform. The modularity allows for easier upgrades and maintenance. The contracts also emit events to provide transparency and allow for easier tracking of state changes of the protocol.

To illustrate their interplay, we can consider the `borrow` operation:

- *`LendingPool.executeBorrow`*`(...)`
  - `amountInEth =` *`PriceOracle.getAmountInEth`*`(...)`
  - *`ValidationLogic.validateBorrow`*`(amountInEth, ... )`
    - `check reserve configuration`
    - `(healthFactor, ... ) = GenericLogic.calculateUserAccountData(...)`
      - `for each user asset, get price in ETH + reserve LTV +`

liquidationTreshold (LT)

· calculate the weighted average of LTV + LT

· calculate the total user collateral and debt in ETH before the borrow

· calculate the health factor before the borrow

* check health factor is sufficient

* check collateral covers amount (per LTV)

* check specific borrowing conditions to prevent abuses (only borrow a portion of overall liquidity, and not from same asset as collateral)

– *ReserveLogic.updateState*()

* update LiquidityIndex, accruing interest since last time

– mint the debt tokens

– *ReserveLogic.updateInterestRates*

* calculate new interest rates based on the updated reserve parameters (utilization of assets, etc.)

It can be seen that each smart contract library contains specialized logic that is needed to fully validate and change the state of the protocol's contracts.

5.2.1 Test suite

The test suite can be found under `test/`.
The `PoolConfiguration.t.sol` tests the setup and configuration of the lending pool contracts. It comprises:

- `setup_pool()`: This function gets a `LendingPoolAddressesProvider` instance for the `Polygon` market and registers it with the main protocol registry. It configures the ownership of the provider, initializes a proxy instance of the `LendingPool`, and saves the proxy address in the provider. It repeats the same process for the `LendingPoolConfigurator`.
- `setup_tokens()`: This function initializes all necessary token instances (`AToken`, `DebtToken`, etc) for both the `NCT` and `MOSS` assets.
- `setup_interestRateStrategy()`: This function initializes the parameters determining the interest rate strategy in `DefaultReserveInterestRateStrategy`.
- `setup_oracle()`: This function gets an instance of the mock Oracle and sets fixed asset prices within it.
- `setup_carbonReserve()`: This function calls the `LendingPoolConfigurator` to initialize the reserves and configure them for borrowing and lending. For usage as collateral, this includes specifying the `LTV` and `liquidityThreshold` at this point.

The `PoolAction.t.sol` contract tests the main actions that can be performed in the lending pool. It comprises:

- `test_deposit()`: This function tests the `deposit()` function of the lending pool. It checks that the correct `Deposit` event was emitted. It asserts that the depositor's account holds the correct amount of newly minted `AToken` afterward. It also tests depositing on behalf of another user and asserts the balance change.
- `test_borrow()`: This function tests the `borrow()` function of the lending pool by simulating the user borrowing from the `LendingPool`. It checks that this emitted the correct `Borrow` event, and asserts that the user holds the correct amount of `DebtToken` afterward.
- `test_repay()`: This function tests the `repay()` function of the lending pool by simulating the user borrowing from the `LendingPool` and repaying the loan. It checks that this emitted the correct `Repay` event and that the full loan was repaid.
- `test_withdraw()`: This function tests the `withdraw()` function of the lending pool. It checks the emitted event and the balance change of `AToken` for the user as well as their holding of the carbon assets.
- `test_transferSharesAndWithdraw()`: This function tests the process of transferring shares and withdrawing assets from the lending pool as a new user. It first deposits carbon assets into the pool and then transfers `AToken` to another user. It checks that the balance of the recipient user is updated correctly. It also tests the process of borrowing against the transferred `AToken` and repaying the borrowed amount, as well as withdrawing and claiming the underlying carbon assets.
- `test_interestRateCalculation()`: This function tests the calculation of interest rates in the lending pool. It first deposits carbon assets into the pool and checks that the initial `liquidityIndex` and other metrics are set correctly. It then simulates the passage of time and triggers the recalculation of the `liquidityIndex`. It checks that the `liquidityIndex` is updated correctly and that the correct amount of interest is accrued. It also checks that the balance of `AToken` in the user's account is updated correctly to reflect the accrued interest.

These tests ensure that the main actions in the lending pool are working correctly and that the state of the LendingPool, including the token balances for the users, are updated correctly after each action. They use the `vm.expectEmit()` function of `forge` to check that the correct events are emitted, and the `vm.startPrank()` and `vm.stopPrank()` functions to simulate actions by different users.

5.3 Vote Escrow Token Contract

This section describes the implementation of the vote escrow contract found at `contracts/protocol/governance/VotingEscrow.sol`.

5.3.1 Overview of the Logic

This contract allows users to lock their GREEN tokens to gain voting power. Voting power is represented by a struct called `Point`, and defined as follows:

```
1     // represent users' voting power at a point in time
2     struct Point {
3         int128 bias; // The accumulated voting power at
              that point in time
4         int128 slope; // rate of change of voting power at
              time
5         uint256 ts; // timestamp of point
6         uint256 blk; // block number
7     }
```

Source code 5.1: Solidity struct to represent the voting power of a user

`slope` and `bias` change both when a user locks GREEN, and when the lock expires. The time is rounded to weeks to simplify the calculations in the contract.

The following `LockActions` are available for the user:

```
1     enum LockAction {
2         CREATE_LOCK,
3         INCREASE_LOCK_AMOUNT,
4         INCREASE_LOCK_TIME
5     }
```

Source code 5.2: User actions for the `VotingEscrow` contract

A lock can be modified in time as well as amount. The longer (or more) tokens are locked, the higher the calculated voting power. Given a lock with `amount` tokens and `duration` locking duration, voting power at the time of locking is given by:

```
bias = slope * duration
```

where `slope = amount / MAXTIME, MAXTIME = 4 years`. It is equation 3.3

$$W(t) = -amount * \frac{t}{t_{max}} + \frac{amount*duration}{t_{max}} = \text{for } 0 <= t <= t_{max}$$

from Chapter 3 using $t = 0$. The struct `Point` is used to store both the global voting power history in `pointHistory`, and the user-specific history in `userPointHistory`. To retrieve a user's voting power at a time `ts`, the contract can access the `lastPoint` in the `userPointHistory` and calculate the `bias` by

```
bias = lastPoint.bias - lastPoint.slope * (ts - lastPoint.timestamp)
```

normalizing it to `0` if the calculation results in a negative value, which represents the case where the user's lock is expired at time `ts`.

Global Voting Power The global voting power is recalculated at each `LockAction` that is initiated by any user. Using this approach the contract always has an updated intermediary value in storage which it can access to retrieve the global voting power, instead of having to iterate over all users first. It also improves the UX by not needing the user to check-in periodically. The contract also changes the global `slope` to account for the expiry of individual user's locks. It uses the `slopeChanges` mapping to store the `slope` changes that are scheduled at a timestamp `ts`. The global `bias` decreases by the updated `slope` at `ts`. [32]
The contract takes a snapshot (checkpoint) of voting power in weekly increments to calculate global voting power. This is described in the next section.

5.3.2 Calculation of Checkpoints

When the user initiates a `LockAction` the following happens:

1. First, the new `slope` and `bias` are calculated to account for the new or extended voting power gained by the new lock. Based on whether there was a pre-existing lock for the user, the old lock's voting power (`bias`) is calculated using the remaining time. The results are stored in local variables as user-specific `Point` structs.
2. The global snapshots (checkpoints) are calculated. The calculation happens in weekly increments (`epoch`) from the last point stored in `pointHistory` to the current timestamp (at which the `LockAction` is happening).
   The loop iterates over the epochs, and subtracts the last point's `slope` from the `bias`. This represents the global voting power decreasing as described by equation 3.3. The slope remains the same. Each intermediary calculation is stored as a new checkpoint in the global `pointHistory` array at the incremented `epoch` index using an estimated block number.
3. After the `epoch` has passed the current timestamp, the global `slope` is updated by any slope changes scheduled for the current time. To reflect the user's `LockAction`, the newly calculated global `bias` and `slope` are now updated by the user's `bias` and `slope` (accessing the results from the first part). Then they are stored at the current `epoch` in the global `pointHistory`.
4. If the `LockAction` was changing an existing lock's duration or amount, it modifies any scheduled slopes in `slopeChanges`. Finally, it records the user-specific checkpoint in `userPointHistory`. This guarantees `slopeChanges`, global and user-specific voting history are up-to-date:
   - If the user changes the amount of an existing lock, the user's voting power (`bias` and `lock`) is recalculated and stored in their history. After step 3 the global history also contains a checkpoint with the updated values, so that any

calculations happening for future epochs will account for the updated lock's effect. The scheduled slope change of the user lock is updated in value.

- If the user changes the duration of an existing lock, a similar thing happens. Their `bias` is recalculated and stored in their history. The global history and the time of the slope change of their lock are also updated.

It is possible to trigger the global checkpoint calculation without a `LockAction` using the `checkpoint()` function.

CHAPTER 6

Results

6.1 Lending Protocol

This thesis has developed a proof of concept for a carbon-asset backed lending protocol called "Green DeFi" with vote escrow tokenomics. The protocol based the design of its lending and borrowing mechanism on the "Aave V2" protocol. This allows the creation of liquidity pools that contain different assets, each in its own reserve, that can be used for lending or borrowing, including carbon tokens. The pools facilitate lending and borrowing without requiring a central order book. Users can deposit assets into the pool, and receive interest on these assets, or use them as collateral in lending. To ensure the trustworthiness of debtors, the protocol requires users to supply collateral of higher value than what they are taking out as a loan. This over-collateralization is at the core of the lending mechanism. Risk parameters, such as LTV, liquidation threshold, or Health Factor, measure and ensure the health of the user's collateralized lending position. If the user is at risk of defaulting, such as when the value of the user's supplied collateral falls too much, the protocol allows external liquidators to pay off the user's debt for a profit. The liquidator receives a part of the user's collateral as a bonus. The risk parameters are determined individually for each asset and depend on its price volatility.

The implementation of the "Green DeFi" protocol (found under Appendix A) contains the whole logic of the lending and borrowing mechanisms. It is possible to borrow and lend, as well as repay and withdraw (carbon) assets again. The protocol ensures that the user's lending position fulfills the necessary collateralization (risk) parameters. These are defined when the reserve is configured after deploying the contracts. The Foundry tests demonstrate a working protocol that is locally deployed and configured to take carbon assets, and allow the lending and borrowing of those.

6.2 Tokenomics

The tokenomics system of the protocol presents a novel approach to acquiring liquidity while encouraging long-term commitment by users. It represents a protocol-owned liquidity approach. Users who provide liquidity receive GREEN rewards. Users gain the ability to influence decisions about these rewards by staking their GREEN and LP tokens. The `VotingEscrow` contract used for the calculation of voting power is at the center of this system. Its abstract requirements were derived in Chapter 3, and its implementation was described in Chapter 5.

The vote escrow contract was implemented and thoroughly tested. The contract behaved as expected for various scenarios, ranging from single and multiple lock creation to lock extension, withdrawal of assets, or triggering global checkpoints. The contract also reverted execution when attempting to perform operations that were invalid, such as increasing a lock to a total duration more than the maximum one. The contract allows the user to create a lock and gain voting power by staking GREEN assets for a certain time (Requirement 2). It is invalid to withdraw assets earlier. Therefore users cannot vote twice with the same GREEN (Requirement 1). The contract calculates the bias and slope and calculates the checkpoints. Users who lock more tokens or lock them for longer can receive more voting power (Requirement 3). Voting power is immediately realized and stored (Requirement 4). The contract accesses an individual user's voting power history and decreases their bias to calculate the voting power over time (Requirement 6). Finally, voting power is represented by a new, non-transferable token called "veGREEN" which represents the GREEN assets that were staked for voting power in the escrow contract (Requirement 5).
Overall, the contract is designed as a fair voting system that cannot be abused and to encourage user engagement. This acts as an incentive for users to engage with the governance and vote, instead of letting their voting power decay, aligning GREEN token emissions with user participation in governance.

CHAPTER 7

Conclusion

This chapter discusses the results of the thesis and the limitations of the work that was conducted.

7.1 Discussion

This thesis attempted to develop a decentralized lending and borrowing protocol that integrated carbon tokens as collateral and implemented a vote escrow tokenomics system to acquire liquidity and incentivize long-term user engagement. It developed a basic proof of concept for this lending protocol. It also derived the requirements of the vote escrow contract, and implemented and thoroughly tested it.
The results are a proof of concept that demonstrates the lending and borrowing of carbon assets on a dedicated platform in DeFi. As the smart contracts were only deployed and tested locally, the results can not be evaluated in terms of "real-world" outcomes. However, if the protocol were to be deployed, there are some possibilities: If bootstrapping enough liquidity and attracting enough users, it could contribute to the adoption and circulation of carbon tokens. As the first carbon-asset focused financial services platform in DeFi, it can also establish carbon tokens as a new asset class in DeFi with unique properties (such as carbon offsetting as a value-added), further helping them to gain traction.

7.2 Limitations

As mentioned, it remains to be seen how the protocol will perform in the "real world" and whether the voting escrow contract can indeed help to bootstrap the protocol's liquidity. Blockchain technology is a multidisciplinary field, and various disciplines are important for its understanding, such as governance, economics, game theory, distributed systems, etc. This makes the protocol design much more complex than what is possible

to investigate here given the time constraints of the thesis.

To further test the protocol and that the risk parameters are working correctly, there would need to be more test cases that reflect real-world usage. However, it is not trivial to construct such tests since the values have to be worked out in advance. It is also not possible to test actual user behavior, as trading and financial markets are governed by the collective behavior of the participants.

7.3 Future Work

The results can serve as a foundation to further develop a lending and borrowing platform that integrates carbon assets and increases the adoption of carbon tokens. Such a protocol can provide more carbon-assets related functionalities, such as paying interest in carbon tokens or integrating a mechanism to offset carbon emissions. DeFi evolves rapidly and as protocols introduce novel concepts and mechanisms for transacting and operating financial services on-chain, carbon assets could become more attractive and widespread with the right incentives.

Bibliography

[1] O. Oyebanji, "Tokenisation of real world assets: An interview with opentrade ceo david sutter," Jul 2023.

[2] Flowcarbon, "Introduction to carbon markets," Nov 2022.

[3] "Putting carbon markets to work on the path to net zero," Oct 2021.

[4] Flowcarbon, "Flowcarbon docs - context."

[5] E. Khodai, "Tokenization of carbon credits: A deep dive," May 2023.

[6] Tal, "Defi lending concepts part 1: Lending and borrowing," Mar 2023.

[7] Aave, "Risk parameters," Sep 2023.

[8] Tal, "Defi lending concepts part 2: Liquidations," Apr 2023.

[9] R. Stevens, "What is tokenomics and why is it important?," Nov 2022.

[10] CoinMarketCap, "Max supply definition: Coinmarketcap," Dec 2022.

[11] E. Lacapra, "Crypto token supplies explained: Circulating, maximum and total supply," Nov 20022.

[12] Melis, "Bitcoin and its purchasing power: an explanation," Oct 2018.

[13] ethereum.org, "How the merge impacted eth supply," 2023.

[14] A. Krishnakumar, "Ethereum as a deflationary asset, explained," Mar 2023.

[15] J. Chung, "Defi 2.0: An alternative solution to liquidity mining," Nov 2021.

[16] Y. Kim, "The curve ($crv) wars and what it means for the curve ecosystem," Jan 2022.

[17] W. Vermaak, "What is yield farming?: Coinmarketcap," Dec 2021.

[18] C. Allred, "What is protocol owned liquidity? olympus dao explained," Apr 2022.

[19] 0xWailord, "A liquidity bootstrapping guide for new daos and defi projects," Jan 2023.

[20] M. Mihajlović, "What is liquidity mining?," May 2023.

[21] Chainlink, "What is defi 2.0?," May 2023.

[22] Nansen, “All hail masterchef: Analysing yield farming activity,” Jun 2021.

[23] Nansen, “Showing the distribution of the number of days that capital entering "masterchef.sol" contract or forks remained in the contract,” Jun 2021.

[24] OlympusDAO, “Olympus - ohm is smart money,” Oct. 2023.

[25] Kryptzo, “Chickenbonds - a method to bootstrap liquidity for early stage projects,” Feb 2023.

[26] CurveDAO, “curve-contract,” Feb. 2021.

[27] L. Guo, “Vote-escrow protocol diagrams on miro,” Sep 2023.

[28] KlimaDAO, “Introducing klimadao,” 2023.

[29] M. Carpenter-Arevalo, “What is klimadao? a deep dive.,” Jun 2022.

[30] Soliditylang, “Contract abi specification,” 2023.

[31] ethereum.org, “Erc-20 token standard,” May 2023.

[32] CurveDAO, “Curve dao: Vote-escrowed crv - curve 1.0.0,” Mar 2021.

APPENDIX A

Source Code

The source code for the Green DeFi protocol can be found here: [SRC] https://github.com/happyhackerbird/green-defi