“Native futures”

Native (perpetual) futures (without a funding-rate).

A sketch of a system to enable liquid markets for the price of ether in the future. The system is perpetual in that futures do not have to expire. There is no requirement for a system to assign a funding-rate as in current perpetual futures markets.

TLDR: Lock ether to mint a range of ERC20 tokens (1:1) that enforce a delay upon unlocking. Trade the tokens freely as an expression of future price prediction.

This is an invitation for comment and criticism. Please share any existing similar work that comes to mind.

Background

- Futures markets increase the efficiency of markets
- Current off-chain futures markets rely on legacy systems that hold user keys
- Some on-chain futures markets in design make use of oracles
- Some have staking tokens that incentivise risk minimisation of the system

Inspiration

- With smart contracts, you can ensure the delivery of capital occurs at a time in the future.
- Every free market participant is an oracle, in that their action brings information about the world into the market. A system that uses every participant to create price information is stronger than a system with special dedicated oracles.
- Free agents can audit a system of smart contracts. The simpler the mechanism, the easier the risk assessment.

Goal

- Create a continuous futures market that integrates with existing infrastructure
- Mechanisms include locking ether for later redemption after a time delay
- The time preference of users, and their perception of future prices will create a market with a differential between ether locked for different durations
- There is no time-critical nature to the market. Futures are always redeemable with a delay that is user-initiated and non-compulsory.
- No oracle

Architecture

- Base contract accepts ether and locks it, returning ERC20 token
- User can return ERC20 token to redeem ether 1:1
- Base contract will delay the redemption of ether by a fixed delay, depending on the ERC20 token
- When locking ether, the contract can issue a range of ERC20 tokens that correspond to different delays. When the ERC20 is returned, the contract will delay the delivery of ETH by the corresponding number of days from the moment of return. The proposed set of ERC20 contracts is:
- 1 day (ETH001DAY)
- 2 day (ETH002DAY)
- 3 day (ETH003DAY)
- 4 day (ETH004DAY)
- 5 day (ETH005DAY)
- 6 day (ETH006DAY)
- 7 day (ETH007DAY)

- 14 day (ETH014DAY)
- 28 day (ETH028DAY)
- 90 day (ETH090DAY)
- 180 day (ETH180DAY)
- 270 day (ETH270DAY)
- 360 day (ETH360DAY)
- 720 day (ETH720DAY)
- 1 day (ETH001DAY)
- 2 day (ETH002DAY)
- 3 day (ETH003DAY)
- 4 day (ETH004DAY)
- 5 day (ETH005DAY)
- 6 day (ETH006DAY)
- 7 day (ETH007DAY)
- 14 day (ETH014DAY)
- 28 day (ETH028DAY)
- 90 day (ETH090DAY)
- 180 day (ETH180DAY)
- 270 day (ETH270DAY)
- 360 day (ETH360DAY)
- 720 day (ETH720DAY)
- When locking ether and minting delayed ether tokens, the user submits the quantity of each token desired, where the sum is equal to the ether provided (e.g. 1 ETH is locked to mint 0.5 ETH005DAY and 0.5 ETH014DAY).
- In its simplest form, a user may lock 1 ETH and mint 1 ETH360DAY. They may then transfer this to another person who can immediately submit the 1 ETH360DAY to the base contract. 360 days later that person will receive 1 ETH.
- It is up to the market to decide what each token is worth at any point up until it is returned to the base contract for destruction and redemption for ether.

Time delayed withdrawal mechanism

Time delay is an approximation based on block speeds. This is an obvious pain point. A prolonged period of delayed or accelerated blocks would cause ether to be available earlier or later than expected.

Delayed-ether is be sent to the base contract along with an address where the ether will be redeemed to. The contract has a hard coded value for the average block time. The contract registers the quantity of ether, the address and calculates the block height after which the ether may be withdrawn.

The contract must then be called after the delay has passed so that the ether is sent from the contract. Two ways this could go ahead: individual calls for individual balances; or anyone can call the contract at any time to release all funds that have passed their delay duration.

Market incentives

A person sees a market where they disagree about the future price of ether. Depending on the situation they may:

- Lock ether, mint delayed-ether then sell it in that market
- Buy the delayed-ether in the market and return it to the contract, receiving ether upon completion of the delay.

Use cases

Believe ether price will be higher at future time

- Buy delayed-ether (at a discount from bears), send to base contract, wait for ether to arrive.

Believe ether price will be lower at future time

- Deposit ether to mint delayed-ether, sell in market at a premium.

Believe ether price will be volatile during future period

- Deposit ether and mint a range of delayed-ether. Provide liquidity in markets where people are trading delayed-ether.

Believe ether price will not be volatile during future period

- I am unsure of how agents would utilise the system to their advantage here.

Scaling

The contract may benefit from scalability solutions that other markets develop and deploy. Because these futures never expire, traders could exchange delayed-ether in layer 2 markets indefinitely. Cheap layer 2 markets could lead to more trades and more smooth price-finding.

Having >10 individual ERC20 contracts as part of the system seems unavoidable. Can you see a simpler arrangement?

Security

The system operates without reliance on external actors to maintain the platform. If the system falls into disuse, every single token may be redeemed for ether. The cost incurred by users in this instance is the opportunity cost of locked capital.

For a contract with few moving parts, the security lies in the ease of auditing the contract.

At deployment, the contract contains the whitelisted addresses of the deployed ERC20 ETHxxxDAY contracts. No additional contracts can be added after deployment.

Each ERC20 ETHxxxDAY contract will only mint in response to requests from the whitelisted base contract.

Trading costs

The following steps incur costs

- Deposit ether to mint delayed ether
- Transfer or trade delayed-ether (standard ERC20 costs)
- Sending delayed-ether to layer 2 (optional with unknown costs)
- Returning delayed-ether to unlock ether
- Making a contract call to withdraw ether

Upgradability

Not possible. Deployment of a separate contract and users would exit the old system and enter the new system.

Referral field

This is an idea that I wish I had seen for other major projects. Including this function in the base contract can provide an example for the broader community to play with, as a solution for other projects.

The concept is to provide a reward for systems to be built around the contract.

When interacting directly with the contract, there is no fee by default. A fee can be triggered during the contract call by the caller, for example a front-end for the contract. The contract would respond by taking a small percentage (e.g. ~0.01%) of the ether entering (or leaving) the contract.

Calling the referral field enables a front-end service to meet development and maintenance costs. The referral fee, if invoked, could also be divided into audit and deployer funds.

Audit funds

None by default.

Optional: The deployer of the original base contract could specify an address of a known public auditor for a fraction of the opt-in referral fee. Once the amount reaches a viable threshold, the auditor could withdraw the funds and perform and audit. (Or withdraw the funds and not perform and audit, but lose credibility. Or withdraw the funds and send them to a different auditor who is willing to do the work.) If an auditor address is in the base contract, but there has not been sufficient volume to perform and audit, interested parties could either do an audit as a public good, or donate funds so that an audit becomes viable.

Deployer reward

None by default.

Optional: The deployer of the original base contract could specify an address for a fraction of the opt-in referral fee. Given that this project is not likely to be very large, this is more about demonstrating ways that other developers could create systems that sustain the creators, and prevent capture by venture capital. In an ideal world venture capitalists are designing and deploying beautiful systems that are so useful that they can make money by taking a small fee as compensation.

Synthetics built from delayed-ether

As an ERC20, the delayed-ether may be wrapped in a variety of ways to create custom financial instruments. Because there is no counterparty risk, wrapped assets do not have to worry about solvency in the underlying delayed-ether system.

Leveraged trading

There is no provision for leverage, however people may create leveraged positions elsewhere to then interact with this system. Importantly, the use of leverage does not introduce risk to the system. This is because the base contract accepts ether, not assets that are created from leverage that are vulnerable to collapse.

Flash loans

No significant risk is foreseen. The architecture of a flash loan involving delayed-ether is as follows:

1. Transaction begins
2. Leverage acquisition
3. Trade for ether
4. Send ether to base contract
5. Mint delayed-ether
6. Dispose of delayed-ether in market
7. Other trading activity
8. No further interaction with delayed ether
9. Leverage is unwound
10. Transaction ends
11. A third party, now in possession of delayed-ether may trade it or redeem for ether.

Step 6 lowers the price of delayed-ether. This would disrupt any contract that uses the spot market price for delayed ether as an oracle. Mitigation can be achieved by those contracts using average price feeds.

Broader impact

If the system becomes widely used, the future price of ether can be taken from a range of markets where delayed-ether is trading. Other systems can use these markets as oracles. Individuals can hedge risk with systems that they can examine and reason about and can avoid taking counterparty risk.

Limitations

In general this feels like a naive approach. Are there game theoretic flaws? Are there enough incentives for participants? Is there a better term for a financial instrument like this?

Many features could be added, but there is a high preference for simplicity.

What do you think?