The Kowala Protocol: A Family of Distributed, Self-Regulating, Asset-Tracking Cryptocurrencies

Eiland Glover and John W. Reitano

Abstract

Cryptocurrencies such as Bitcoin, Ether, and Dash exhibit significant volatility. Consumers, merchants, traders, investors, miners and developers have a need for a cryptocurrency whose value can be counted on to remain roughly stable from one day to the next and whose operation does not depend on potentially unreliable third parties such as banks.

Cryptocurrencies with automated value-pegging mechanisms, such as NuBits and BitUSD, have suffered repeated failures due to malfunction and lack of adoption, while companies that hold centralized, one-to-one reserves in fiat, such as Tether, have proven vulnerable to the whims of banks and governments.

The *Kowala Protocol* is our proposed method for creating a new family of cryptocurrencies which maintain stable values while retaining other benefits of cryptocurrencies, such as decentralization, security, privacy, speed of transfer, and low transaction costs.

Why A Stable Cryptocurrency Is Needed

Beyond the considerable benefits that most cryptocurrencies bring, a stable cryptocurrency will provide unique benefits to many stakeholders, such as consumers, merchants, investors, traders, miners and developers.

Some use cases of a stable cryptocurrency for consumers are:

- avoiding cryptocurrency volatility when purchasing real-world products and services;
- avoiding volatility of less stable fiat currencies (such as of those of Venezuela, Zimbabwe, Nigeria, etc)
- providing a stable store of value; and
- gaining access to bank-like services offered by third parties (for consumers without easy access to bank accounts denominated in stable fiat currencies).

Some use cases for merchants are:

- avoiding volatility when selling products and services for cryptocurrency; and
- gaining access to bank-like services offered by third parties (for merchants without easy access to bank accounts denominated in stable fiat currencies).

Some use cases for investors are:

- parking funds in a decentralized, stable asset.

Some use cases for traders are:

- pursuing arbitrage opportunities in new cryptocurrency markets;
- parking funds in a decentralized asset that has stable value; and
- trade in and out of stable asset at low cost and high speed.

Some use cases for miners are:

- pursuing mining profits without investing in expensive hardware and electricity costs;
- reaping mining rewards consistently, no matter whose machine solves the block; and
- having the option to sell or lease acquired mining rights.

Some use cases for developers are:

- incorporating payment functionality into apps and websites without the need to establish a merchant account; and
- incorporating such payment functionality while avoiding cryptocurrency volatility.

What is the Kowala Protocol?

The Kowala Protocol defines a method for constructing a family of distributed, self-regulating, asset-tracking cryptocurrencies called *kCoins*.

Each kCoin is designed to be traded on open exchanges and to maintain a close to one-to-one value relative to any widely traded asset such as a currency (USD, EUR, JPY, etc.) or other asset. Each kCoin is identified by a symbol consisting of the letter *k* followed by the symbol of the underlying asset. For example, the kCoin of USD is kUSD, that of EUR is kEUR, and so forth.

kCoins constantly gather market information from endorsed sources and regulate their value through three core mechanisms: minting variable amounts of coins, applying variable fees, and obtaining feedback from an active and well-informed trading market. In time, these mechanisms always return each kCoin to parity with its underlying, tracked asset. The certainty of each kCoin's eventual return to parity, in turn, creates pure arbitrage opportunities for traders seeking to profit from slight fluctuations in kCoin market prices around the peg.

We chose the codebase of the Go Ethereum client[1] as the starting point for the development of kCoins in order to access both Ethereum's strong feature set

[1] See https://github.com/ethereum/go-ethereum

(especially its sophisticated smart contract facilities) and the collective abilities and ongoing efforts of its development team. On top of this foundation, each kCoin adds the value stabilization and market observation features described above. Finally, to achieve ultra-fast transaction processing performance, we have replaced the Ethereum consensus protocol with Konsensus, an implementation of PBFT derived from Tendermint. Because every kCoin needs a robust exchange market to function properly, each kCoin is implemented as a distinct, independent blockchain with its own tokens, smart contracts, mining community, etc.

Note: The discussion below applies to any kCoin, but, for ease of explanation, we will use the specific example of kUSD—the kCoin pegged to the U.S. dollar—in most of the remainder of this white paper.

Key Components of the Kowala Protocol

At the center of the Kowala Protocol are three stability-related mechanisms that are designed to keep the market price of kUSD at or very near $1.

The section *Stability Mechanism 1: Minting Algorithm* below describes how a variable amount newly minted coins is used to push the market price toward the target of $1 when necessary.

The section *Stability Mechanism 2: Stability Fee* describes how, in the scenario in which the market price is below $1 and a low minted amount is not sufficient to bring it back to $1, a special, variable fee is applied to each subsequent transaction until the market price begins to rise.

The section *Stability Mechanism 3: Trading Activity* describes how traders, informed by the previous two mechanisms and motivated by a desire for profit, are expected to engage in trading activity that accelerates the return of the market price of kUSD to $1.

Beyond these stability mechanisms, the other principal components of the Kowala Protocol are a unique consensus protocol (see section *Konsensus*), a system for reporting prices via oracles (see section *Price Oracles*), and rewards for users who hold balances in a way that benefits the network (see section *Stability Rewards and Stability Contracts*.

Note that all aspects of the Kowala Protocol described in this white paper are subject to change.

Stability Mechanism 1: Minting Algorithm

To calculate the amount of new coins to be minted, we will need to introduce four new concepts.

The first concept is a *dead-end address*, which is a address that is precluded from having outbound transactions and thus whose balance can increase but not decrease.

The second concept is the *available coin supply*, which is the total value of all coins that are available for sending via transactions. Formally this is defined with the following function:

$$availableCoinSupply(b) := totalCoinSupply(b) - \sum_{d \in D(b)} balance(d, b)$$

where *totalCoinSupply(b)* refers to the total number of coins issued as of block number *b*, *D(b)* is the set of all dead-end addresses as of block *b*, and *balance(d, b)* is the balance of dead-end address *d* as of block *b*. In other words, the available coin supply is the total number of sendable coins.

The third concept is the *minted amount cap*, *cap(b)*, which is defined as:

$$cap(b) := \begin{cases} 0.0001 \times availableCoins(b-1) & b > 1, availableCoins(b-1) \geq 1000000 \\ C & otherwise \end{cases}$$

where C is a fixed initial cap that applies when the available coin supply is very low (under 1 million coins). Initially, we set C to 82, since this achieved good results in our testing, but this constant can be changed in future updates to the protocol.
The minted amount cap will be used to ensure that, after an initial growth period, the newly minted amount will never be larger than one basis point of the entire available coin supply.

The fourth concept is the *market price of* kUSD, *p(b)*, which is determined by a system of oracles which is designed to come to a trustworthy and accurate consensus on the value of kUSD as reported publicly on exchanges (see the section *Price Oracles* below)

With all these concepts in place, we are ready to define the amount of new coins to be minted, *mintedAmount(b)*, as:

$$mintedAmount(b) := \begin{cases} 42 & b = 1 & (initial) \\ min(A \times mintedAmount(b-1), cap(b)) & b > 1, p(b) \geq p(b-1) > 1 & (high - price) \\ max(\frac{mintedAmount(b-1)}{A}, 0.000001) & otherwise & (normal) \end{cases}$$

where A is a block-by-block adjustment factor. Initially, we set A to 1.0001, since this achieved good results in our testing, but this constant can be changed in future updates to the protocol.

The calculation of *mintedAmount(b)* is split into three scenarios: *initial*, *high-price, and normal.*

During the *initial scenario*, which applies only to the first block, the minted amount is set to the arbitrary value of 42.

Next, we consider the *high-price scenario*, which occurs when, over the course of two consecutive blocks, the price of kUSD is over $1 and is rising or flat. In this scenario, the minted amount is set to a small increase (based on the adjustment factor) over the previous block's minted amount. This increase is subject to the minted amount cap.

We initially hypothesized that, when large numbers of newly minted coins are earned by miners, a large portion of such coins will reach exchanges as market sell orders and drive down the price of kUSD. A detailed agent-based behavior model with multiple market scenarios supports this hypothesis (see *Agent-Based Modeling* below). For this reason, we posit that for the high-price scenario, no further mechanism is needed to reduce the price to $1.

Finally, we consider what we call the *normal scenario*. This scenario occurs when the price of kUSD is either (a) less than or equal to $1 or (b) over $1 and falling. In this scenario, we do not want to mint additional kUSD since this would likely reduce

the price, which is at best unnecessary and at worst harmful. For this reason, the normal-scenario portion of the function states that we should gradually reduce the minted amount by dividing the prior minted amount by 1.01 (subject to a minimum of 0.000001 kUSD).

Repeated applications of the formula in the normal scenario during a prolonged drop in price can quickly lower the minted amount to nearly zero. For example, in this scenario, reducing the minted amount from 1 kUSD to 0.000001 kUSD takes fewer than 1400 blocks (less than 24 minutes, assuming a block time of 1 second). However, even a near-zero minted amount may not be sufficient to raise the price of the coin if there is a large drop in coin demand during the same period. In the section *Stability Mechanism 2: Stability Fee* below, we will address this insufficiency by introducing a way to materially reduce the total coin supply.

Stability Mechanism 2: Stability Fee

Like Ethereum. the Kowala Protocol stipulates that every transaction sender be charged a fee, called the *transaction fee*, which is used to maintain the network. However, the transaction fee in kUSD does more than compensate miners for their computational effort (measured in *gas*, an execution fee levied on every operation). It also includes an additional fee, called the *stability fee*, which ranges between 0 and 2%) of the transaction amount. The primary purpose of the stability fee is to raise the price of kUSD by reducing coin supply when the price of kUSD is below $1. The following formula shows the two parts that comprise the transaction fee:

$$transactionFee(t) := computeFee(t) + stabilityFee(t)$$

where *computeFee(t)* refers to the compute fee (see section *Compute Fee* below) and *stabilityFee(t)* refers to the stability fee, which will be defined later in this section.

The stability fee should conform to the following constraints:

- under normal condition, the stability fee is zero; and
- in the abnormal condition of a prolonged drop in demand for kUSD, the stability fee should still represent no more than 2% of the transaction amount, and should be lowered back down to zero once normal conditions

return.

In real-time uses of a payment system, we will also need to define a formula for an *estimated stability fee,* which should conform to the following constraints:

- it is easy to calculate; and
- it is either equal to or just slightly above the actual fee, assuming the actual transaction takes place within 15 minutes of calculating the estimate

The stability fee is initially set to the same value as the compute fee. This ensures that transactions that impose more computational costs on the network will bear more of the costs of maintaining stability. Then we will periodically increase it by a small amount (9%) until it rises to its maximum value (2% of the transaction amount).

For the purpose of defining the stability fee and estimated stability fee formally, we will introduce a few functions.

First, we define the price change rate, *p'(b)*, as:

$$p'(b) := \frac{901 \sum_{i \in N(b)} p(i)i - \sum_{i \in N(b)} p(i) \sum_{i \in N(b)} i}{901 \sum_{i \in N(b)} i^2 - (\sum_{i \in N(b)} i)^2}$$

where *N(b)* is the set of integers from *b-900* to *b*.

Next, we introduce the function repeat(b), which is used to used to track how many times we need to increment the stability fee:

$$repeat(b) := \begin{cases} -1 & b = 1 \\ repeat(b-1) & \text{b mod } 900 \neq 1 \\ repeat(b-1)+1 & b > 1 \ \& \ b \ mod \ 900 = 1 \ \& \\ & p(b) < 1 \ \& \ p'(b) \leq 0 \\ max(repeat(b-1)-1, -1) & otherwise \end{cases}$$

The definition of *repeat(b)* implies that the stability fee changes at most once every

900 blocks (15 minutes, assuming a block time of 1 second). A value of -1 is used by this function to signify that the stability fee should be 0.

With these new items defined, we are now ready to define the *stability fee* and *estimated stability fee* for transaction *t*:

$$stabilityFee(t) := \begin{cases} 0 & repeat(block(t)) = -1 \\ min(1.09^{repeat(block(t))} computeFee(t), 0.02 amount(t)) & otherwise \end{cases}$$

$$estimatedStabilityFee(t) := min(1.09 * stabilityFee(t), 0.02 amount(t))$$

where *block(t)* and *amount(t)* are the block and amount, respectively, associated with transaction *t*. See *Compute Fee*, below, for a definition of *computeFee(t)*.

Once received from the transaction sender, the stability fee will be split into two equal parts; one half will be burned (i.e., transferred to a dead-end address), and the other half will be distributed to certain holders of kUSD in a way that encourages them to hold increased balances at key times. See the sections *Coin Burning* and *Stability Rewards and Stability Contracts* below for details on how these quantities are handled.

Our modeling shows that a non-zero stability fee is applied infrequently in anticipated market conditions, and is effective in raising the price of kUSD when applied (see *Agent-Based Modeling* below).

Stability Mechanism 3: Trading Activity

Given sufficient time, the first two mechanisms above will cause the price of kUSD to revert to parity (i.e., 1 USD per kUSD). We label this tendency to revert to parity the *first-order effect* of the Kowala Protocol. Once the first-order effect has become recognized among market participants, we then expect three *second-order effects* to come into play.

First, since the parity price is the only particular price to which kUSD has a natural inclination to return, it constitutes a game-theoretic focal (or Schelling) point. The absence of perfect communication and trust among disparate human market

participants along with the status of the parity price as a focal point increase the likelihood that the price of kUSD will return to parity.[2]

Second, whenever deviations of the price from parity do occur, they will give rise to short-term profit opportunities for professional arbitrageurs. The source of these opportunities is the difference between the time horizon of relatively patient arbitrageurs and that of other market participants for whom the short-term need to move into or out of kUSD exceeds concern over these price deviations. Although arbitrageurs may exploit these profit opportunities purely for self-interest, their trading activity has the positive effect of accelerating the return to parity.

Third, confidence in the reversion to parity will strengthen further as participants observe a history of such reversion in the marketplace.

All of these second-order effects depend on the existence of a well-functioning USD/kUSD (or equivalent BTC/kUSD) exchange market. Since the existence of such exchange markets are in the interest of exchanges, miners, and users of kUSD, it is highly likely that they will arise through the self-interested activities of these groups.

For its part, Kowala plans to participate in independent, open-market, profit-seeking trading activities based on the same information publicly available to all.

Compute Fee

The *compute fee* is the portion of the transaction fee which is transferred to the miners of kUSD and is determined by the following formula:

$$computeFee(t) := gasUsed(t)gasPrice(t)$$

where *gasUsed(t)* is the gas used in processing transaction *t*, and *gasPrice(t)* is the price of one unit of gas at the time of transaction t. Both gasUsed(t) and gasPrice(t) are defined similarly to their analogs in Ethereum, and the compute fee itself is similar Ethereum's "transaction fee". For simple send transactions, gasUsed(t) is set to 21,000, and, for smart contract transactions, its value will depend on the

[2] See https://en.wikipedia.org/wiki/Focal_point_(game_theory)

operations triggered by the associated smart contract. The default value of *gasPrice(t)* is currently set to *2 x 10*$^{-8}$.
The sum of compute fees for all transactions in the current block will be awarded to the elected proposer of the block:

$$computeReward(b) := \sum_{t \in T(b)} computeFee(b, t)$$

Coin Burning

The burn amount for a block is defined as half of the sum of all transaction stability fees for a given block:

$$burnAmount(b) := \frac{\sum_{t \in T(b)} stabilityFee(t)}{2}$$

where T(b) is the set of all transactions that form block b.

The burn amount is sent to a dead-end address, which has the effect of permanently reducing the available supply of kUSD. Over time, the burn amount allows kUSD to respond effectively to a decrease in demand. For example, if just 15% per day (on average) of the kUSD coin supply is transacted via on-chain transactions, the burn amount will reduce the coin supply by over 8% in 30 days and over 41% in 180 days.

Stability Rewards and Stability Contracts

The purpose of stability rewards is twofold: to provide access to network ownership to an expanding circle of people and to incentivize behavior that supports network utility (by reducing the quantity of kUSD available for sale when the price of kUSD is below $1).

The distribution of stability rewards to kUSD balance holders is managed via special-purpose smart contracts called *stability contracts*, which are implemented as child contracts of a single master contract. Each stability contract stores multiple addresses: a *source address*, a *savings address,* and a *stability reward address*. The

source address refers to the original address from which funds are transferred to a savings address. The private keys for the remaining stored addresses are derived deterministically from the private key of the source address, thus giving the source address owner the exclusive ability to move funds out of these other addresses, subject to certain constraints described below.

To receive a stability reward, a source address owner must have previously created a stability contract for their source address and have moved some kUSD from their source address into the contract's savings address. Additionally, a transaction constraint prevents transactions from a savings address from being sent anywhere other than back to the source address.

As a preparation for defining the stability reward amount, we first define the stability amount as the half of the total stability fees for the block. Note that this amount is equal to the burn amount (see *Coin Burning* above). In other words, half of the total stability fees are directed to the burn amount and the other half to the stability amount:

$$stabilityAmount(b) := \frac{\sum_{t \in T(b)} stabilityFee(t)}{2}$$

The stability amount is then partitioned into individual *stability rewards*, one for each stability contract, using the following formula:

$$stabilityReward(b, a) := \left(\frac{savingsBalance(b, a)}{\sum_{x \in A(b)} savingsBalance(b, x)} \right) stabilityAmount(b)$$

where *b* is a block number, *a* is a source address with an associated stability contract, *A(b)* is the set of all source addresses associated with stability contracts as of block b, and *savingsBalance(b,x)* denotes the balance as of block b of the savings address associated with the smart contract with source address x during block b.

The stability reward for each smart contract is transferred into the associated stability reward address, and a second transaction constraint prevents this stability reward from being transferred out until the price of kUSD returns to $1 or higher. While an owner may remove funds from his savings address at any time, he can only

claim the funds in the stability reward address once the price of kUSD returns to $1 or higher. If he removes the savings address funds before this time, the associated stability reward funds will be forfeited.

mUSD and Other mTokens

The Kowala Protocol is a two-token system. The primary token in a Kowala Protocol blockchain is a kCoin, that is, a stable-value payment token such as kUSD. In addition, associated with each kCoin is a secondary, mining token or *mToken*; for example, the mToken associated with kUSD is called *mUSD*. Proof of ownership of a minimum number of these mTokens (currently 30,000) is a requirement for mining the associated kCoin.

Each mToken will be associated with cryptographic addresses and other meta-information used to securely manage a) tracking of performance of mining responsibilities, b) the recipient address for mining rewards, c) mToken ownership, and b) delegation of mining responsibilities to third parties. A specialized smart contract will be created to manage this information.

There will be 2^{30} (1,073,741,824) mTokens minted for each kCoin. Once the initial mTokens have been minted, a smart contract feature will prevent additional mTokens from being created in the future.

Konsensus: The Kowala Consensus Protocol

The consensus protocol of Kowala, *Konsensus*, is a formal set of rules that allows a decentralized group of participants to agree on the advancement of the blockchain, including the handling of transactions and the distribution of incentivising rewards.

Konsensus is derived from Tendermint, a mostly asynchronous consensus protocol which is itself based on Byzantine Fault Tolerance (BFT).[3] We chose to base our work on Tendermint because it has achieved block transaction performance that is

[3] Much of the Tendermint and PBFT summary below is drawn from the following 4 sources:
PBFT White Paper: http://pmg.csail.mit.edu/papers/osdi99.pdf
Cosmos White Paper: https://github.com/cosmos/cosmos/blob/master/WHITEPAPER.md#consensus
Tendermint Overview: https://github.com/tendermint/tendermint/wiki/Byzantine-Consensus-Algorithm
Jepsen Review of Tendermint: https://jepsen.io/analyses/tendermint-0-10-2

far superior to other widely used blockchain consensus protocols and because it has a well-understood security profile. Like Tendermint, Konsensus is a network protocol that requires its participants to be in constant contact with each other.

In Konsensus, a network is comprised of two or more nodes connected by a well-defined communications protocol. A node, in this context, is a computer running a Kowala Protocol-compatible mining client. Communication is achieved using the standard Ethereum model of TCP and UDP.

Like other cryptocurrencies, kUSD is mined. For every block, all qualifying miners have the chance to earn a reward of kUSD by contributing to the advancement and security of the blockchain. A miner is an active node on the network that controls a minimum number (currently 30,000) mTokens. In order to advance the blockchain, the network must include at least one miner, though we would typically expect there to be many more.

Mining in Konsensus proceeds one block at a time. For every block, a leader, called the *proposer*, is deterministically elected from the current miners. All other miners are known as *validators* for the duration of the block. The function of the proposer is to assemble and propose the next block in the chain, and the function of the validators is to vote on whether to accept the proposed block or reject it. The proposer first assembles a viable block that includes pending transactions and then distributes the block to validators for formal verification. Each validator inspects the block for suitability and then casts a cryptographically-signed vote to accept or reject the block. If two-thirds or more of the validators vote to accept the block, the proposer must commit it to the blockchain and signal to the entire network that the block is finalized. If the two-thirds threshold is not reached, the block is rejected. The process then begins again, with a new proposer elected to propose another block.

Election Based on mToken Ownership

Miners do not have an even chance of being elected as the proposer; rather, their likelihood of being chosen is proportional to the number of mTokens associated with the node. The likelihood of the election of a given miner is given by:

$$electionChance(m, b) = \frac{tokensHeld(m, b)}{\sum_{x \in M(b)} tokensHeld(x, b)}$$

where *m and M(b)* are a particular miner and the set of all miners, respectively, who are mining during the current block b. Over time, a miner will be elected proposer with a frequency proportional to the ratio of the number of tokens he or she holds to the total tokens held by all active miners. After being elected, a proposer must correctly perform the actual work of proposing, and will be then rewarded for this work.

Price Oracles

The valuation of the price of kUSD is achieved through a system of price *oracles*. In this system, each participant is incentivized to determine the correct price of kUSD based on publicly available sources, and to report this price periodically to the kUSD network by sending an *oracle stake* to a special smart contract. The participants whose prices are within one sigma of the median will receive rewards minted for this purpose. This approach incentivizes participants to discover and report a trustworthy and accurate consensus value of the market price of kUSD.

As each block is generated, a small amount of the newly minted amount, called the *oracle deduction*, is placed into a specialized smart contract, called the *oracle fund*, and these accumulated fees are paid out to oracles as rewards when they report prices. The oracle deduction is determined by the following formula:

$$oracleDeduction(b) := oracleDeductionFraction(b) mintedAmount(b)$$

The function *oracleDeductionFraction(b)* is not defined here but will be return values between 0 and 0.01 in such a way that, in normal circumstances, the oracle fund will have a balance sufficient to pay out anticipated rewards to oracles.

We use the term *minted reward* to describe the the portion of the minted amount that remains to be given to proposers after the oracle deduction is deducted. The amount of the minted reward is simply:

$$mintedReward(b) := mintedAmount(b) - oracleDeduction(b)$$

Normally, the reward to oracles is set equal to 4% of the total minted reward over the last 900 blocks (approximately 15 minutes):

$$baseOracleReward(b) := 0.04 \sum_{b-900<x\leq b} mintedReward(x)$$

However, no reward is paid to oracles unless the proposer is qualified as an oracle during block b. To be qualified as an oracle, a proposer must maintain a minimum stake of mUSD (currently 6 million) and not be banned by a majority of other oracles. This reward is further limited by the actual balance in the oracle fund:

$$payableOracleReward(b) := min(baseOracleReward(b), fundBalance(b))$$

Here is the complete formula for the reward, which we term the *oracle reward*:

$$oracleReward(m, b) := \begin{cases} payableOracleReward(b) & m\, is\, an\, oracle \\ 0 & otherwise \end{cases}$$

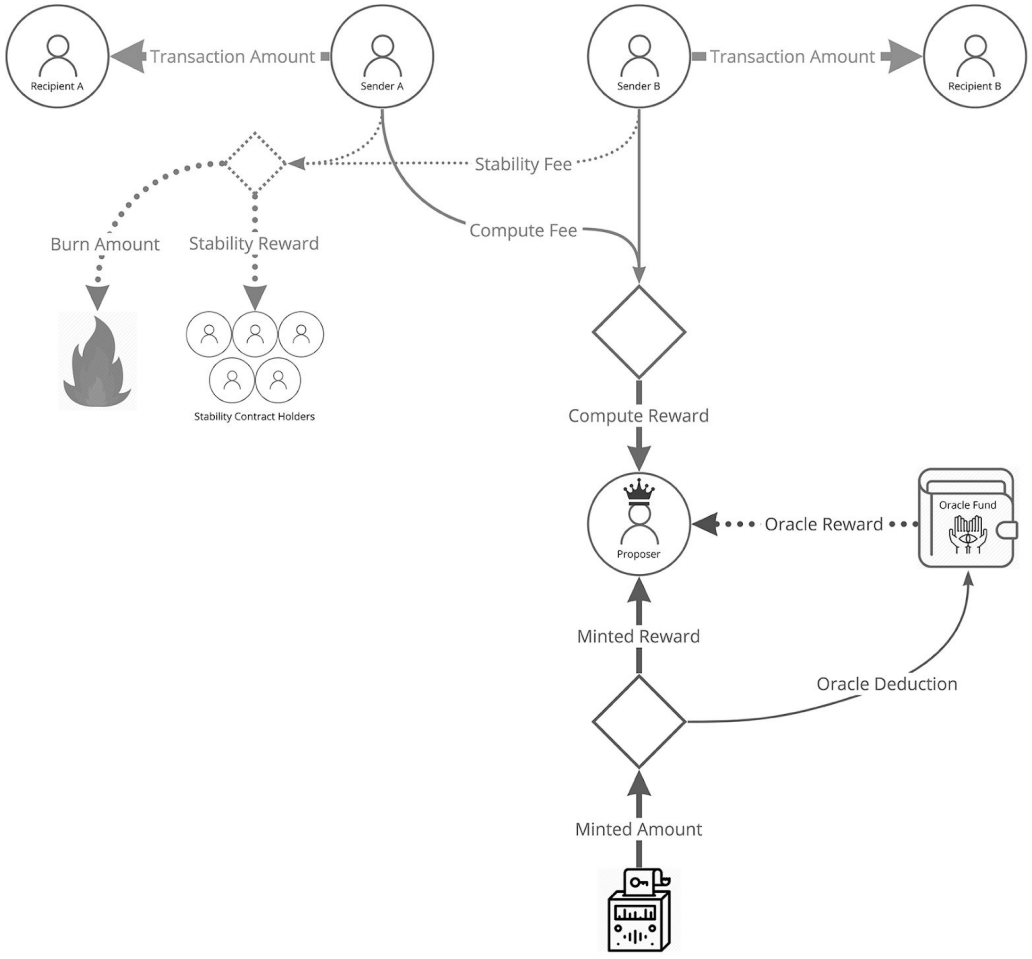
Mining Reward

In Konsensus, the crucial work is done by proposers. The proposer's role requires significantly more resource and energy expenditure than the validators'. The proposer must interrogate the blockchain to determine the amount to be minted (see *Stability Mechanism 1: Minting Algorithm*, above), assemble the block and ensure it is distributed properly—all in a short timeframe—which is non-trivial. Indeed, the processing of blocks would halt without a proposer, but can continue if any number of validators (or even all of them) leave the network.

The total reward earned by the proposer for his or her work is called the *mining reward*, and is equal to the sum of three components: 1) the minted reward , 2) any applicable oracle reward, and 3) the compute reward:

$$miningReward(m,b) := \begin{array}{r} mintedReward(b) + \\ oracleReward(m,b) + \\ computeReward(b) \end{array}$$

where m is a miner and b is a block number. The following diagram shows these three components, and the source of their respective values:



Proposer Eligibility, Proposal Stake

Konsensus punishes dishonest or free-loading miners by making them ineligible to propose blocks (and thus earn rewards) for a game-theoretically sufficient number of blocks. For example, a would-be proposer must have participated in the prior 10,000 blocks as a validator in order to be eligible to propose the current block.

In order to receive mining rewards, proposers are required stake a minimum number of mUSD (currently 30,000). This is done by authorizing one or more entries in a special-purpose smart contract set up for tracking mining activity. The mUSD owner will separately specify the addresses that will receive the kUSD rewards associated with these mUSD, and these reward receipt addresses are independent of the node performing the associated mining.

The amount staked is labeled a *proposal stake*. Normally, the proposal stake is reclaimed after the proposal is accepted or rejected, but if the proposal is determined to be dishonest, the stake is forfeited.

Transaction Speed

In Konsensus, there is no concept of post-creation confirmation. Blocks are irreversible once created. Compare this to Ethereum, where 6 or more 15-second block cycles may need to pass before a block is considered non-reversible.

We are actively investigating optimization of Konsensus, with the goal of processing 7,000+ transactions per second with a typical per-transaction processing time of one second. This performance would compete favorably with real-world commercial payment systems such as Visa.

Reduced Energy Usage

Because the Konsensus avoids the need for the expensive search for hash solutions, there is no incentive for miners to use powerful, energy-wasting mining hardware to outperform their fellow miners. This approach provides security by incentivizing the distribution of blockchain validation-by-consensus across many independent parties (because only active miners receive validation rewards) but uses very little

electricity compared to typical cryptocurrency consensus mechanisms. For example, one analyst estimates that there are between 5,000 and 100,000 Bitcoin miners who collectively consume approximately 774 megawatts of electricity.[4] By comparison, we estimate that 100,000 miners could run kUSD client software on low-power machines and consume less than 5 megawatts of electricity.

Continuous Agent-Based Modeling

We have created a sophisticated agent-based software model to test the simultaneous use of all three mechanisms that form the Kowala Protocol. We have run a significant number of simulations against various permutations of the model, including:

- variations in constants used by the Minting Algorithm
- variations in constants used to define the Stability Fee
- variations in starting conditions
- market demand fluctuations, including mass panics
- rapidly increasing and decreasing numbers of participants
- excessive optimism and pessimism of arbitrageurs and prospectors

The algorithms and constants described in this white paper reflect the results of our testing, and the Kowala Protocol incorporates only the behaviors which consistently and reproducibly yielded the best results in our models.

Future agent-based modeling will be based on actual historical market data. Kowala will regularly create large-scale simulations of established marketplace behavior and conduct predictive research into further refinements to the Kowala Protocol. These enhanced simulations will feature agents whose behavior is derived from genetic algorithms informed by actual, historic market decisions. Kowala will also model potential attacks by malicious actors in order to preempt them and to uncover other unforeseen vulnerabilities.

[4] See https://bravenewcoin.com/news/number-of-bitcoin-miners-far-higher-than-popular-estimates

Current Research Area: PID Controllers

Our approach of continuously incorporating market feedback into our coin-supply adjustments is broadly reminiscent of the concept of a PID-controller, a sophisticated engineering model of goal-seeking action coupled with continuous feedback from a sensor. We are currently investigating the possibility of improving our mechanisms by expressing them as PID-controllers.[5]

Current Research Area: Artificial Intelligence

We are currently investigating the incorporation of Artificial Intelligence into the Kowala Protocol in two areas. First, we are investigating the use of deep learning to improve the responsiveness of the minting and burning rates of the stability mechanisms. Second, we are investigating the use of AI to discourage or defeat adversarial and exploitative trading behavior.

Conclusion

This white paper has identified the problem of volatility in cryptocurrency and proposed the Kowala Protocol as a robust solution to this problem. Although we have established through extensive modelling that the protocol works in many anticipated scenarios, more work is needed to demonstrate with higher certainty that the specific mechanisms described here will work in a real-world market. We invite others to contribute to improving the Kowala Protocol by visiting https://kowala.tech and to participate in the development of the kUSD client software located at https://github.com/kowala-tech/kcoin.

[5] https://en.wikipedia.org/wiki/PID_controller,
http://www.eurotherm.com/temperature-control/principles-of-pid-control-and-tuning, and
http://www.eurotherm.com/temperature-control/pid-control-made-easy