Kosu: A Decentralized Liquididty Aggregation Primitive

Liam Kovatch, Henry Harder
www.kosu.io

April 22, 2019

Abstract

We motivate Kosu, a decentralized network and protocol that encourages the curation of a consistent, high-quality, verifiable, public order book. The Kosu network is an independent bonded proof-of-stake blockchain built on Tendermint Consensus. The lightweight state application is responsible for enforcing a simple access control ruleset and order booking procedure. The network utilizes a shared security model, wherein all staking and token-based mechanics happen on the Ethereum main-network. The model provides unidirectional communication and checkpoints between the two networks by leveraging Tendermint to provide finality for specific Ethereum-based state changes. Individuals intending to post orders to the Kosu network are required to bond tokens for the duration they wish to have access. The poster bonding mechanism creates a market for the allocation of network resources and an implicit incentive to contribute quality liquidity to the network. Kosu validators are selected via a token curated registry system on Ethereum. This registry system allows both the inflation rate as well as the number of active validators to be market driven and determined by the network's stakeholders. The network is highly generalizable, supporting orders of arbitrary maker-taker based virtual settlement systems. In its entirety, the proposed system effectively curates a decentralized order book that serves as a liquidity aggregation primitive for second layer systems. Kosu will be free to use and open source, reducing barriers to entry for both developers and market agents.

Contents

1 Introduction

Historically, financial markets have been relatively siloed. To date, centralized entities control significant entry points and accounting systems, limiting access beyond a small set of privileged participants [10]. The recent invention of blockchains, decentralized ledger systems, is now challenging this trusted paradigm. Ethereum, a public blockchain with a built-in Turing-complete programming language, provides developers an immutable and verifiable computation platform for scripts known as "smart-contracts". These scripts can encode arbitrary, deterministic state transition functions, providing a powerful primitive for more complex ledger based application systems [20].

Since the network's genesis in 2014, developers have leveraged smart contracts for a wide range of novel protocol systems. Perhaps the most adopted technical standard for smart contracts is the ERC-20 token implementation [11]. The standard defines a common token interface allowing for the creation of a diverse set of Ethereum-based assets. These tokens often serve as a critical crypto economic primitive providing core utility for protocol systems also deployed on Ethereum. As such, a system of exchange is a key requisite for a token-dependent contract ecosystem.

In early 2017, the 0x Project published a whitepaper defining a maker-taker based exchange protocol in which a pipeline of publicly accessible Ethereum smart contracts facilitate the wallet-to-wallet settlement of funds. In this model, order broadcast and discovery happen off-chain. Independent entities known as relayers are responsible for sourcing and maintaining liquidity in the form of independent collections of signed order messages. This hybrid architecture drastically improved the efficiency of existing DEX systems while also providing a critical open exchange primitive with high extensibility [1]. Since the first deployment of the 0x protocol, a significant number of projects have adopted it as a common piece of infrastructure facilitating 713,000 trades and $750,000,000 in volume at the time of writing. Moving forward, the team continues to innovate on the scalability, efficiency and incentivization of DEX systems.

Virtual settlement systems, specifically decentralized exchanges (DEXs), are a relatively accessible use case for blockchains; a direct evolution from the original digital currency use-case. While conceptually apparent, the architecture of DEXs is a complex and evolving subject. The scalability restraints of blockchain systems combined with the complexities of an asynchronous discrete time system limit their compatibility with optimally efficient continuous time order books and settlement systems [2]. As such, alternative DEX architectures have also captured considerable market share. These systems include dutch auctions and automated market making systems. While immediately compelling for some use cases, the evolution towards hybrid decentralized exchange structures, similar to 0x, is likely. These systems, though currently constrained, have a clear path towards scalability and offer the most scalable, efficient and extensible DEX structure to date [23].

The Kosu network is designed within the 0x-motivated ecosystem, providing a decentralized aggregation primitive for arbitrary limit orders. More specifically, the Kosu network aims to provide a highly extensible decentralized complement to independent centralized aggregators, or relayers. The set of orders maintained on the Kosu network will represent a high-value subset of the available order universe.

2 Overview

Kosu is a public blockchain designed to support the construction of order booking and liquidity systems for emerging decentralized financial primitives. The Kosu network uses a bonded proof-of-stake security model implemented with Tendermint Consensus [3] and a system of Ethereum contracts that include the protocol's native token.

Kosu is primarily composed of two interdependent systems[1] that together implement the protocol's core function: to establish and curate a consistent and verifiable set of recent, high-quality orders. The Kosu contract system is responsible for defining the protocol's incentive mechanisms, facilitating validator curation by stakeholders, and specifying an access control scheme for the Kosu network. The functionality provided by the contract system supports the network-based order booking service on which exchange and liquidity systems can be built.

The Kosu network is a system of globally distributed nodes collectively maintain and update the shared order book[2] and blockchain while providing a one-way peg to Ethereum. The network supports basic order booking functionality, access control for user accounts, and cryptographic proofs of the existence of state contents, including the set of orders that compose the current "order book". Validators[3] secure the network by submitting attestations about specific state changes to the Ethereum contract system, processing orders, and reaching consensus on the canonical chain and overall application state.

The Kosu protocol's Ethereum-based smart-contract system implements the protocol's native token and manages core incentive mechanisms. These mechanisms manage access control for posters and allow stakeholders to curate the Kosu network's validator set through a novel weighted voting system and token-curated registry.

Kosu validators are required to run full Ethereum nodes and are responsible for submitting special attestations, or "witness" transactions, to the Kosu network about specific state changes to the protocol's contract system. Specifically, the network must reflect updates to users' bonded token balances (which affect a order rate limit enforced by the network), and updates to the dynamic registry contract containing the curated list of validators.

Individuals who wish to leverage the network's decentralized order booking and message relay features are called posters, and can gain write access to the network by bonding any amount of Kosu tokens in a poster registry contract. After their bond transaction is confirmed on Ethereum to a certain depth, validators update the Kosu network's shared state to reflect the bonded balance change of the new poster. The same mechanism allows posters to adjust the amount of tokens they have bonded, or withdraw entirely from the system at any time. The amount of tokens a poster chooses to bond determines the maximum number of orders they may post per defined period, which is proportional to their bonded balance relative to the total number of poster bonded tokens.

At deterministic intervals based on the height of the Ethereum blockchain, Kosu validators compute a simple rate-limit mapping based on in-state balance data that allocates network order throughput proportionally to posters who have Kosu tokens bonded at the height a new interval starts. These intervals are called rebalance periods and allow the relatively continuous process of balance updates on the Ethereum Poster Registry, as a result of bonding/unbonding by posters, to be mapped to more discrete periods of rate-limiting in a deterministic manner. At the beginning of each new rebalance period, validators must reach consensus on the parameterization of the upcoming period. The starting and ending Ethereum block heights of the period, and the total number of orders to be accepted from posters during that time is determined prior to allowing posters write access for the period.

During normal network operations between rebalance periods, validators accept and process incoming order transactions from posters (gossiped via connected full nodes), while submitting regular attestations (called witness transactions) to other validators on the network. This process ensures the state of the Kosu Ethereum contract system is accurately tracked.

[1] These systems are described in more detail in section 3.

[2] The Kosu order book is a set of hashes of specific recent orders, from which conventional limit order books can be constructed. See section 3.3.4.

[3] Kosu non-validating full nodes crucially provide access to the order book and enable users to submit orders, laying the foundation for a wide array of applications. Full nodes also allow a wide variety of proofs to be constructed about the existence or absence of state data.

The validity of order transactions is based on the signature of the poster that submitted the order to the network, and a certain required data structure that enables recovery of the signature. Validity is defined as orders containing valid recoverable signatures from a poster account with a non-zero remaining limit for the current rebalance period. Validators will include such orders in a block and decrement the posters bandwidth allocation limit for that period for each order submitted.

The Kosu network uses a bonded proof-of-stake security model wherein validators stake (by locking) tokens into a contract proportional to the amount of vote power they wish to receive. The number of tokens they associate with their stake are locked for the duration the entity wishes to validate. The validator may be stripped of their power at any time at the discretion of voting token holders, and if voted out, their tokens are distributed to the individuals responsible for raising – and voting in – the successful challenge.

During the continuous validator governance and curation process, individuals wishing to become validators submit proposals in which they specify a positive or negative reward schedule. If this value is positive, the validator will be rewarded newly minted tokens that inflate the existing supply. If the value is negative, validators will be required to constantly collateralize their listing at the rate specified so their tokens may be burned. In this case, the negative "reward" deflates the existing supply.

The ability for validators to specify both negative and positive reward rates is a crucial component of the overall system design. For the nascent network, validators will be able to extract little value from the act of validating alone, thus will require inflationary rewards that justify the costs of managing validator infrastructure. However, in a mature state the Kosu network could present implicit value capture opportunities to validators in the form of positional information advantages. More generally, validator reward schedules are market-driven and account for both implicit and explicit reward capture.

2.1 Usage example

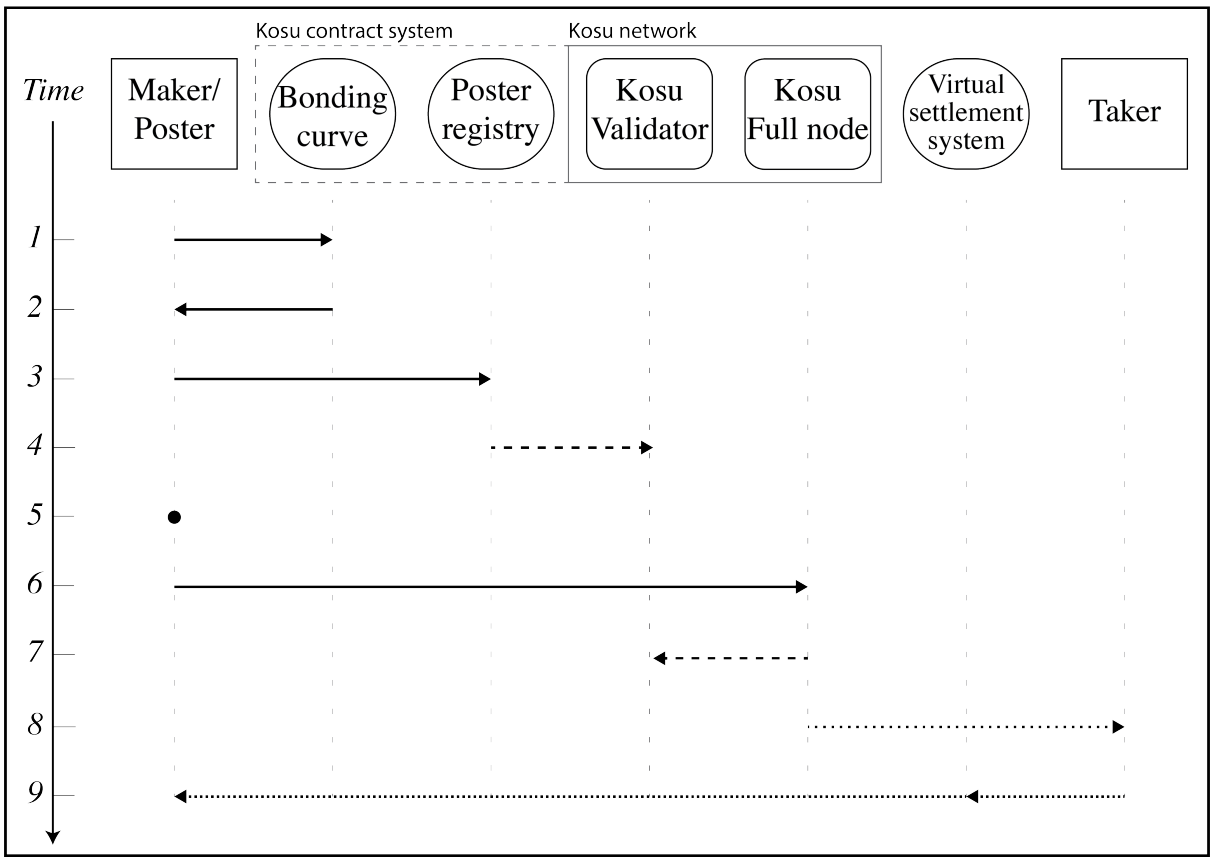

Figure 1: Simplified chronology (arranged with time progressing in the downward direction) of the steps involved for a new user (the poster) to submit an order to the Kosu network.

Figure 1 illustrates the actors, systems, and procedures involved in enabling a new user (the "poster") to contribute liquidity, via a signed order message, to the Kosu network and order book.

Boxes represent individuals or human-controlled actors, ovals represent Ethereum smart-contracts, and boxes with rounded corners represent Kosu nodes. Solid arrows and dots indicate actions triggered by the poster, dashed arrows represent interactions validators are responsible for, dotted arrows represent actions triggered by the "taker", the individual or entity who fills the poster's order by submitting it to a virtual settlement system (such as the 0x protocol [1]).

The individual time-steps are shown on the left (and annotated below), with the first action taking place at time-step 1.

1. A poster who intents to contribute liquidity to the Kosu network "buys" KOSU tokens by locking an arbitrary amount of Ether in the token's bonding curve contract.
2. In the bond transaction, the contract "sends" KOSU tokens to the poster (by increasing the corresponding balance entry in the token contract) according to the bonding curve equation and it's current parameterization.
3. The poster bonds any amount of their new KOSU tokens in the Poster Registry contract
4. The Kosu/Ethereum peg-zone updates the poster's balance in the Kosu network's shared state, granting them write access to the order book.
5. The poster creates and signs a maker order message, indicating their intent and ability to trade.
6. The poster submits their order message as a transaction to the Kosu network via a full node (or light client).
7. A Kosu validator includes the order transaction in a block and makes the order available to the greater network.
8. A taker (any individual filling maker order messages by submitting them for settlement) becomes aware of the new order from their own full node or light client.
9. The taker submits the maker order and any additional required information to an arbitrary blockchain-based virtual settlement system (such as 0x) according to the maker order's type. In most cases, the settlement system will initiate some type of transfer of funds between the maker and the taker.

3 Specification

The specification for each system in this section serves to communicate the purpose and functionality of each component, rather than provide a fully accurate description of specific implementation details.

Many nuanced components and processes are referenced at a high level for the sake of brevity and digestibility. Each component of the protocol's implementation is open source and should be referred to for the most detailed and up-to-date specification and implementation [17].

3.1 Orders

The Kosu protocol defines a simple and extensible data structure to represent signed order messages on the network. The primary purpose of the defined order schematic is to allow signature recovery for the verification of poster bandwidth allocation during order processing and validation.

The order format described below is also designed to allow the "wrapping" of existing hybrid-decentralized[4] order messages (such as 0x [1] and Dharma [16]) for relay on the Kosu network, and settlement through a system of generalizable forwarding contracts.

Usage of the forwarding contract system is strictly optional, and unrelated to the core protocol in the sense that no state is shared between the protocol contract system and settlement contracts (called SubContracts, defined below).

3.1.1 SubContract interface

Kosu can act as an order message aggregator and/or transport layer for a variety of types of on-chain exchange systems. To achieve this, a simple and extensible contract-based interface is defined that allows a common order message format to be used for a variety of Ethereum settlement pipelines that leverage hybrid decentralized exchange architecture.

All current SubContract implementations are in Solidity and designed for use with Ethereum, however, any language and blockchain that supports sufficient scripting capabilities to satisfy the interface above can be used with Kosu[5].

Method name	Returns	Params.	Description
`makerArguments`	`string`	-	Returns a JSON string containing a data structure that specifies the required fields for maker orders for the SubContract.
`takerArguments`	`string`	-	Returns a JSON string containing a data structure (identical to `makerArguments`) that specifies the values a taker must supply when filling a maker order.
`isValid`	`boolean`	`makerData`	Checks weather a maker order is valid and fillable based on the SubContract's validation implementation
`amountRemaining`	`uint256`	`makerData`	For settlement types that support it, this method can return information that allows partial fills
`participate`	`boolean`	`makerData`, `takerData`	The main settlement logic implementation for SubContracts, which triggers execution of a trade by the taker submitting the maker data and their counterparty information.

Table 1: Describes the SubContract interface and method signatures in a language-independent manner.

Some of the data structures used by the `SubContract` interface and Kosu order message structure are described in more detail below. The full interface definition in Solidity can be found on the `ParadigmFoundation` GitHub [19].

[4] Refers to DEX architecture where asset settlement takes place on-chain, and order matching/relay occurs elsewhere.
[5] Posters must provide RLP-encoded SECP256K1 signatures in orders submissions regardless of settlement type.

- `makerArguments`: a structure that defines the name and types of each argument in the `makerValues`
- `takerArguments`: defines the arguments (name and type) takers must provide to `participate`
- `makerData`: an array of 32 byte slices of serialized data included in the maker order message
- `takerData`: an array of 32 byte slices of serialized data provided by the taker as arguments

Since validity conditions of a given order vary greatly between settlement systems, a generic validation interface method (`isValid`) is provided. A method that returns an arbitrary integer (`amountRemaining`) is also specified, which can be used to support partial fills for settlement types where it is logical, such as spot exchange implementations.

3.1.2 Order message format

A simple order message format is defined based on the SubContract interface. The data structure was specifically designed to satisfy the following requirements:

1. Compatibility with new and existing hybrid off-chain settlement systems (0x [1], Dharma [16], etc.)
2. Efficient and lossless compression with existing transports and multiple languages in mind.
3. Support for serialization to EVM compatible data types and structures.
4. Support for arbitrary signature schemes[6] within maker order messages

The Kosu order message format is described below in a language-agnostic manner, however the type annotations indicate the JSON type each top-level field is serialized to. The underlying structure of the non-primitive types (`array` and `object`) are defined in the protocol's implementation [18] and documentation.

Field name	Type	Required	Description
`subContract`	`string`	yes	The deployed address of the target SubContract settlement implementation. Defines expected arguments.
`maker`	`string`	yes	The address of the party that signed the maker order. Usually indicates the beneficiary of settled funds.
`makerArguments`	`array`	no	An array of equal length to the number of `makerArguments` containing objects that define the name and data-type for each argument.
`takerArguments`	`array`	no	Similar to `makerArguments`, it defines the values and data-types required for settlement. Not required for maker orders.
`makerValues`	`object`	yes	A hash-map data structure that contains the parameters necessary for a valid maker order of the target settlement type.
`makerSignature`	`object`	no	An optional field that can be used to include a signature from the maker. May also be included in `makerValues`.
`posterSignature`	`object`	yes	Stores the signature resulting from a poster entity signing a hash of the maker order values. Used to verify poster has bonded tokens.

Table 2: Generic and high-level description of the Kosu order message format, including optional fields.

[6] All orders still must include an SECP256K1 signature from a poster Ethereum account with bonded Kosu tokens.

3.2 Ethereum Contract System

3.2.1 Introduction

The Kosu protocol implementation includes a suite of Ethereum contracts that are responsible for a set of core functionalities related to access control (sybil tolerance), network validator governance, and the protocol's core economic mechanisms (described in detail in section 3.5).

The contract system allows the primary users and stakeholders of the system to participate in governance of the active validator set, manage their Kosu token balances and allowances, and/or gain write access to the Kosu order book.

Specific components of the overall contract system leverage proxy contracts to provide external interfaces to system components, while allowing the actual implementation contract to be replaced if network stakeholders ever deem such action necessary.

The Kosu token[7], the protocol's native asset, is a component of the Ethereum contract system. The token is used throughout the protocol (via the contract system) to allocate resources on the Kosu network (see 3.3 and 3.5), as the asset validators must stake, and to incentivize the curation of a high-quality validator set (see 3.2.6 and 3.5.1).

The contract system is modular and designed with upgradability in mind. Due to the permanence of smart contracts, the ability to deploy upgrades and additions to existing contract systems is limited, unless specific consideration is made during the system's design. Kosu's architecture allows such upgrades and additions to the system (with stakeholder approval[8]) through the use of proxy contracts and a mutable authorization registry contract for managing the system's internal permissions.

3.2.2 Architecture

The Kosu contract system uses a system of public-facing[9] proxy contracts that perform calls to independent implementation contracts where the primary logic and state management resides.

Authorization of internal system methods (including management of the validator set and token supply), is managed by a registry contract that contains the addresses of protocol contracts that are permissioned to access those methods.

Such an architecture is desirable primarily for the following reasons:

1. A component implementation may be updated while the proxy contract's address stays the same[10].
2. Implementation contracts can operate in "trusted environments" by only accepting calls from the designated proxy.

3.2.3 System components

Below is a list of the primary components (contracts) that make up the overall Kosu contract system, and a brief description of the functionality of each. More detail is provided about certain components in the following subsections.

- `AuthorizedAddresses` - a registry containing the addresses of each protocol contract that enables a method modifier to restrict access to internal and sensitive methods.
- `Authorizable` - implements a method modifier that restricts callers to addresses in the authorization registry (`AuthorizedAddresses`), inherited by many protocol contracts.
- `EventEmitter` - a contract responsible for emitting all system event logs. It is used by the Witness component of the Ethereum peg zone (see 3.4).

[7] The KOSU token implements the ERC-20 standard interface [11]
[8] See the future work section for details on the transition to general stakeholder governance.
[9] The proxies are intended to be the interface points with the system, while no meaningful interactions are possible with implementation contracts.
[10] In order for this to be true, there must be no breaking changes to the component's interface.

- `KosuToken` - an implementation of the ERC-20 interface and the protocol's native token.
- `Treasury` - central contract that manages system accounting and allowances, and many other crucial functions related to balances and token supply.
- `PosterRegistryProxy` - the proxy contract for the `PosterRegistry` that provides its external interface.
- `ValidatorRegistryProxy` - the proxy contract for the `ValidatorRegistry` that provides the external interface for validator governance and voting mechanisms.
- `PosterRegistry` - implementation contract that allows users to bond[11] and un-bond KOSU tokens at arbitrary times and amounts in exchange for write access to the Kosu network order book.
- `ValidatorRegistry` - an implementation of a modified token-curated-registry [14] that allows any KOSU token holder to participate in the selection of the Kosu network's validators. Stakeholders are rewarded for participating in governance (for certain outcomes) by challenging new and existing listings, and voting on those challenges.
- `Voting` - implementation contract that manages challenges and votes for the `ValidatorRegistry` contract. Implements a basic commit-reveal voting scheme.

The only contracts that provide meaningful functionality for external users are the Treasury, KOSU token contract, and the registry proxies for posters, validators, and voters.

3.2.4 Treasury

The Treasury contract is responsible for managing system token allowances, executing adjustments to token supply (as a result of the validator reward process), and other functionality related to asset management within the Kosu contract system.

It acts as a central accounting system and tracks the balances of users tokens throughout the system, including those bonded by posters, staked by validators, or idly deposited within the treasury. This accounting system is crucial to the systems voting mechanics, which relies of the Treasury's account of each user's total system balance to assign vote power to participating stakeholders.

3.2.5 Poster bonding

The Poster Registry contract is a simple registry system that allows users to bond (register) and un-bond (release) Kosu tokens at any time, and in arbitrary amounts. The contract maintains a mapping of accounts to bonded token balances, and importantly, triggers the emission of an event log through the EventEmitter contract each time a poster registers or releases tokens, including the account address and the new balance.

Generation of the rate-limit mapping that allocates network order throughput (see 3.3.3) to posters is handled by the Kosu network, after balance updates to the contract system are reflected in the networks state. This process, described in section 3.4, is part of Kosu's Ethereum peg-zone.

3.2.6 Validator token-curated registry

The Validator Registry contract is an implementation of a token-curated registry (TCR), and a cornerstone piece of the Kosu protocol that enables decentralized curation of the networks validator set.

There are three main concepts that behind token-curated registries. The goal of a TCR is to produce a curated set of *listings*, which are entries that have successfully been included in the registry. Listings begin as *proposals* which indicate intent to join the registry, backed by a number of tokens. Proposals and listings may be *challenged* at any time, where a challenger must put up an equal number of tokens as the listing or proposal owner. Challenges trigger votes, which any token holder may participate in.

The specific mechanics of listings, proposals, and challenges within the Kosu Validator Registry are described below.

[11] The term "stake" is avoided here since the balance may not be slashed, and does not provide rewards for lockup.

1. Anyone may create a proposal that indicates their intent to become a network validator
   (a) Proposals must include a stake greater than, or equal to a specified minimum stake.
   (b) Proposals also include a "reward" to be executed to the validator on a periodic basis.
      i. This reward may be positive (tokens minted) or negative (tokens burned).
      ii. If it is negative and the listing is accepted, the validator must continually collateralize a treasury balance sufficient to cover their burn rate.
      iii. If they are unable to cover a burn, they may be removed from the listing without a full challenge ("touch-and-remove")
      iv. If a listing owners system balance falls below their stake size, they may also be touch-and-removed.
   (c) Proposals that are not challenged (see below) are automatically accepted after a certain period of time measured in Ethereum blocks.
2. Pending proposals, as well as accepted listings may be challenged at any time.
   (a) Challengers must have a system balance of tokens equal to or greater than the stake of the listing or proposal being challenged.
   (b) Challenges may be voted on by any Kosu token holder using a commit-reveal vote scheme during a specified voting period.
   (c) Vote weight is determined by the balance of a user within the Kosu contract system.
   (d) After the completion of a vote period, the ruling is determined based the binary outcome that received a majority of the vote weight.
   (e) If a challenge resolves against a listing or proposal, those staked tokens are distributed to the challenger and the winning voters.
   (f) If a challenge fails, the challengers collateral is distributed to the winning voters and the owner of the challenged listing.
   (g) Voters are never penalized for voting on the losing side of a challenge.
   (h) Failed challenges lose their collateral to the challenged listing and the winning voters.

3.3 Tendermint Network

3.3.1 Introduction

The Kosu network is a Byzantine-Fault Tolerant (BFT) bonded proof-of-stake blockchain built on Tendermint Core that is designed to act as a decentralized order message aggregation and curation primitive.

The core state machine, replicated across the network's nodes by Tendermint, is designed with simplicity, efficiency, and safety in mind. It provides the minimal functionality necessary to establish a robust base layer upon which more sophisticated order booking and liquidity systems can be built.

A primary feature of the network is the establishment of a canonical set of recent, valid orders (see 3.1) in each Kosu block. This order set, occasionally referred to as an "order book" due to helpful conceptual parallels, can be used to construct and verify a database of recent, curated order messages. This set of orders is computed deterministically by Kosu validators during block generation, and a hash representing the canonical order set for each block is included in the application's state tree (see section 3.3.4).

The network uses a shared security model, wherein Kosu validators are full Ethereum nodes, and thus must stay in sync with both chains. This model allows the Kosu network to maintain a one-way peg-zone with the Ethereum blockchain, where state changes to the Kosu contract system (balance updates, etc.) can be reflected in the Kosu network's state, by way of special attestation "witness" transactions. Validators vote on and agree to apply state changes that receive support from a supermajority of the networks validating vote power.

Order messages are treated as a finite network resource, constrained by physical bandwidth limitations, the number of transactions that can be included within a block, and the size of the order messages themselves. Order throughput is allocated to posters over discrete intervals called "rebalance periods", which are parameterized by validators based on the height of the Ethereum blockchain, and per-period order limit.

Validators establish the maximum number of orders to accept per rebalance period as a configurable consensus parameter, and proportionally allocate throughput to posters based on the number of Kosu tokens each has bonded. This mechanic acts as a sybil tolerance mechanism, and creates a market for access to order throughput. Individuals can always gain more bandwidth allocation by bonding more tokens in the Poster Registry (see 3.2.5).

Posters, individuals contributing order messages (see 3.5) to the network, sign their orders with the Ethereum keypair they used to bond tokens in the Poster Registry. When validators process these order transactions, the poster's signature is recovered and checked against the in-state mapping of bandwidth allocations. Each valid order from a poster decrements their period limit by one, until a new period begins and the allocation is recomputed.

3.3.2 Architecture

Central to the Kosu protocol is its primary state application, a deterministic finite transactional state machine, replicated across nodes by Tendermint Core and its underlying consensus engine.

The overall architecture of the Kosu network is similar to a conventional Tendermint network, with the addition of the infrastructure necessary to support a one-way peg-zone (see section 3.4) to the Ethereum blockchain. The network is constructed of computing nodes running a client implementation of the Kosu network. Full nodes with Tendermint key-pairs that correspond to a listing in the Validator Registry (see 3.2.6) are designated as validators: they collect transactions and take turns proposing arrangements of the transactions in blocks.

The Kosu network's architecture around a peg to the Ethereum blockchain is comparable to that of a side-chain, but there is an important distinction in that no state is shared from the Kosu network to the Ethereum contract system. The contract system is responsible for establishing the validator set and the poster account-balance mapping, and the network implements specific functionality based on this subsetted state. See section 3.4 for a full description of this one-way peg zone.

The networks shared state is stored in Merkle tree that self-balances using a modification of the AVL algorithm, introduced by Tendermint [9]. Such a structure allows efficient verification about the correctness of a new state via the Merkle root after execution of a block(a requirement for consensus to be reached through Tendermint). Additionally, usage of a Merkle tree allows light-clients and applications to construct and verify proofs about state contents.

Kosu validators must run the following processes (some of which may be available through the same client implementation):

- Ethereum client - all validators must run full Ethereum nodes in order to track the contract system's state, and submit necessary attestation transactions.
- Kosu state machine - the primary application logic, which implements the Tendermint application blockchain interface (ABCI) [5].
- Kosu Witness - the validator process that implements the Kosu/Ethereum peg zone and submits Witness attestation transactions to the state machine.
- Tendermint Core - manages networking and state-machine replication of the Kosu sate machine via the Tendermint Consensus engine.

Validators likely will not run or provide public API or RPC services due to security concerns, therefore non-validating full-nodes play the important role of providing access to the network for most applications, infrastructure, and users.

Due to the fact that validators can be high-value targets for attacks (both for network denial-of-service, and targeted attacks to compromise the highly valuable validating keypairs), Kosu validators are expected to use special proxy full-nodes called "sentries" [8] to access the network, and be isolated from the public internet.

3.3.3 Poster access control

The submission of order transactions is limited over deterministically calculated time intervals called rebalance periods. These periods are parameterized by validators with rebalance transactions (see 3.3.5) that specify the start and end of the period, indicated in Ethereum block heights. The rebalance periods additionally specify a maximum number of order transactions to accept within that period.

During generation of these rebalance transactions, Validators (via their Witness processes) snapshot the most up-to-date bonded balances of each account in the Poster Registry contract (see 3.2.5), and proportionally allocate the number of orders to be accepted that period to posters based on the following equation(s).

// do eqs. in latex

When the Ethereum block marking the end of a rebalance period is finalized[12], each validator will create and sign a rebalance transaction, which includes a proposed rate-limit mapping constructed using the method described above. These transactions are verified based on the correct parameterization of the period (period length, order limit, etc.) as well as the accuracy of the proposed rate-limit mapping.

Rebalance transactions from non-Byzantine validators should include rate-limit proposals that match the one generated from existing in-state balances. Due to the fact that all rebalance transactions from behaving validators should be identical, it does not matter which proposal is accepted. The usual case is the validator that is block proposer when the round-ending Ethereum block is finalized includes their own proposal. All subsequent proposals for that round will fail.

As posters submit order messages (signed with their Ethereum keypair used to bond Kosu tokens), their per-period order limit is decremented until either it reaches zero and no more orders are accepted from that proposer, or a new period starts and their limit is re-computed.

3.3.4 State model

Kosu nodes maintain an application state which includes the balance of poster accounts (updated through the peg-zone and witness mechanisms), the bandwidth allocation for posters (derived from their balance), information specific to the validator set to support its contract-based curation, and the parameterization of rebalance periods.

The applications initial state is introduced in a hard-coded genesis block, and all subsequent states are generated deterministically from previous states, and the transactions processed within each block. Kosu's genesis block will be generated after the contract system is deployed, allowing time for stakeholders to curate an initial validator set. Kosu's genesis block will specify no initial validators. Instead, the genesis state generated from Kosu's Ethereum contract system's state at a specific height will indicate the staked balance of each entry in the Validator Registry, and the Kosu client implementation will deterministically generate and set the genesis validator set during execution of the `InitChain` ABCI method upon initialization of the chain [6].

The state itself is represented in a mutable self-balancing AVL Merkle tree [9], and the Merkle root hash is used to represent the `AppHash` included by Tendermint in the header of each block [5].

Usage of a Merkle tree data-structure enables proofs about the existence, absence, and accuracy of state data, which is a necessity for a network such as Kosu that can benefit from the existence of light-clients (see section 6). It also enables the Kosu state machine to easily determine if it has the correct state for a new block by comparing the Merkle root hash of its own state against the `AppHash` included in each block's header. Usage of an accurate and deterministic calculation, such as a Merkle root, allows Tendermint to safeguard the application from functioning in undefined states [6].

[12] A block is considered finalized when it has reached a sufficient arbitrary age (number of blocks mined above it).

Some of the notable structured maintained in the application's state are described below.

Field name	Description
`Posters`	A mapping of Ethereum addresses to poster accounts that contain a balance of staked tokens, and their current order limit (or quota) remaining for the active rebalance period.
`Validators`	A mapping of Tendermint-specific hashes of validator public keys (node IDs) to validator accounts that contain their associated Ethereum address, number of staked tokens, and fields used to track and record information about each validator.
`Events`	A mapping of Ethereum event data (identified by hashes of the data) populated from WitnessEvent transactions. Acts as a confirmation queue while events wait sufficient confirmation from validator attestations.
`RebalanceInfo`	Stores the parameterization of rebalance rounds, including the starting and ending Ethereum heights, and the maximum number of orders to accept in the period.
`LatestOrders`	A fixed length, first-in-first-out hash set containing the transaction ID's of a parameterizable number of the latest order transactions. This hash-set makes up the stateful component of the network's "order book".

Table 3: Summary of important fields maintained in the network Merkle state trie.

The primary account objects and state fields are described above, however other fields are included related to internal network functionality, and ultimately affect the state's Merkle root inserted in each Tendermint block header.

3.3.5 Transaction types

The Kosu state machine implements four primary state-modifying transaction types, each described in detail in this section. Three of the transaction types are referred to as "internal" transactions, and require a signature from an active validator key-pair. These internal transactions (described below) facilitate the peg-zone to the Ethereum blockchain (see 3.4) and manage access control for posters.

The fourth type of transaction are order transactions that require a signature from a valid poster account, and may be relayed (gossiped) to validators for inclusion in a block from full nodes. Internal transactions must originate from validators.

Message name	Internal	Description
`WitnessEvent`	yes	Primary message that supports the Ethereum peg zone. Submitted by each validator as an attestation to a state change to Kosu's Ethereum contract system. Triggered by updates to poster's bonded balances, and changes to the Validator Registry.
`WitnessRebalance`	yes	Submitted by validators at the end of each rebalance period to parameterize the next period. Includes a proposal for the poster rate-limit mapping which is verified against in-state balances.
`EthCheckpoint`	yes	Submitted by validators on a best-effort basis as an attestation to each new Ethereum block. Does not affect rebalance mechanism or finality implementation, but can be used to verify correct age between finalized and new Ethereum blocks.
`ProposeOrder`	no	The only external (non-validator origin) transaction type, ProposeOrder messages are signed and submitted by bonded Posters to submit orders to the Kosu network.

Table 4: Summary of Kosu network state-modifying transaction message types.

`WitnessEvent` type transactions specify a "subject" field, which corresponds to the subject of the state change the event was generated to reflect. The subject can be set as "posters" for updates to

the Poster Registry and the bonded balances of its accounts, or set to "validator" for updates to the Validator Registry. WitnessEvents require a super-majority of active vote power to attest to a given event (identified by a deterministic hash) before the corresponding modification is made to the primary state, such as the addition or removal of a validator.

`WitnessRebalance` transactions are submitted by validators when a rebalance period ends, marked by the maturation of the Ethereum block included in the period's parameterization as the ending block. The submitted message includes a proposal for the poster rate-limit to be used in the next period. Because this mapping is generated deterministically from in-state balances, all WitnessRebalance proposals for a given period should be identical from non-byzantine validators. Generally the rebalance transaction from the validator who happens to be proposing that block is accepted, given it matches the expected mapping from in-state balances.

`EthCheckpoint` transactions serve to correlate a given Kosu block height with the latest-known (non-matured) Ethereum block on a best-effort basis. Due to the possibility of reorganization of recent Ethereum blocks, the latest-known checkpoint is not used for any consensus critical mechanisms (such as generation of rate-limit mappings) and exists solely for validators to indicate the best-known Ethereum block. Validators submit an EthCheckpoint message each time a new Ethereum block is mined, and once a supermajority of validators attest to that block, it is accepted as the "latest" Ethereum block.

`ProposeOrder` transactions may be submitted by posters who have bonded Kosu tokens in the Poster Registry, and count against that poster's rate-limit for the active rebalance period. Assuming the ProposeOrder message is structurally valid, and the signature is recoverable and originated from a poster account with a remaining limit, the order is accepted. Upon acceptance of an order, the posters limit is decremented, and the order's hash is added to the fixed-length first-in-first-out order hash set included in the network's Merkle root.

3.3.6 Validator curation

A motivating requirement of Kosu's design was the ability to support a highly dynamic proof-of-stake validator set. The incentive model of the overall protocol (see 3.5) requires that protocol stakeholders have the ability to curate the networks validators, both to facilitate healthy competition for the position, and to allow the removal of validators who may not be explicitly byzantine (by lying or attempting to compromise consensus), but who may otherwise be engaging in behavior stakeholders deem implicitly malicious. These considerations motivated the design of the curation mechanism (described in section 3.2.6) and supporting peg-zone (described in 3.4).

Kosu's underlying networking and consensus engine, Tendermint Core, allows a genesis validator set to be hard-coded or computed during network genesis, and for updates to the validator set to be applied by existing validators at the end of each block [6].

To avoid the special case of selecting genesis validators to launch the network with, the Kosu protocol's launch will be multi-phased, with the contract system being deployed ahead of the networks genesis. Deploying the contract system prior to the launch of the network will allow the decentralized curation mechanism to produce an initial listing state, which can be used to compute the initial validator set.

A specific Ethereum block height will be agreed upon at which point node operators will snapshot the contract system's state, and populate Tendermint's genesis file with the initial state. During the one-time execution of the `InitChain` ABCI method [6], the Kosu state application will apply the deterministically-computed genesis validator set and block production will begin. All listings included in the registry at the specified Ethereum block will become the network's first validators, from which point normal network operation and peg-zone functionality will commence.

The next section introduces the Kosu Ethereum peg-zone, and provides an overview of its construction and implementation.

3.4 Ethereum Peg Zone

3.4.1 Introduction

Communication between blockchains, specifically those with differing types of finality guarantees, requires the usage of a peg-zone. The Ethereum blockchain currently offers probabilistic finality for

state-modifying transactions included in blocks based on its proof-of-work consensus mechanism. Older PoW blocks are exponentially harder to reorganize [13] than recently mined blocks, thus as the age of a block increases, the chance of that block or its contents being moved in the blockchains chronology decreases exponentially.

This is in contrast to the security model Kosu uses where economic finality is guaranteed nearly instantly, thanks to the underlying Tendermint Consensus [3, 15] engine. As soon as a super-majority of the network's vote power supports the commit of a given block, it can be considered finalized and irreversible. It would be impossible for any entity to "reorganize" the canonical chain in the proof-of-work sense without possessing a superiority of vote power through the acquisition of a significant number of Kosu tokens, and approval to the validator set by existing token holders.

The Kosu to Ethereum one-way peg-zone facilitates a small subset of Ethereum's state to be represented on the Tendermint network. The usage of a one-way peg allows Kosu to leverage the security and uptime of the Ethereum network for critical extra-protocol actions, such as the transfer and exchange of KOSU tokens, and validator set curation.

Generalized and two-way peg-zones require complex construction [4] and additional security assumptions, so Kosu uses a simple architecture for its peg-zone that is specific to a subset of state changes in the Kosu contract system (see 3.2).

3.4.2 Architecture

The architecture and design of Kosu's one-way peg zone is conceptually simple, and builds off a two-way peg-zone design initially proposed by Cosmos [4]. Kosu's peg zone comprises of two main components, one residing in the Kosu contract system, and the other as part of Kosu's Tendermint-based network.

Kosu's contract system (see 3.2) contains a special "logging" contract called the EventEmitter which emits descriptive event logs for certain state changes to the overall contract system. The Kosu peg-zone implementation is concerned with two types of events emitted by the EventEmitter.

One event type describes an update to the Poster Registry, when a new poster bonds token, or an existing poster modifies their balance by adding or withdrawing tokens. The event contains the Ethereum address of the poster, and an unsigned integer representing their new balance (which may be 0).

The second event describes updates to the Validator Registry, when a new validator is added, or an existing validator is removed. The event contains the Ethereum address to be associated with the validator, the number of Kosu tokens they have put at stake (which directly affects network vote power), as well a 32-byte ED25519 public key to be registered as a validating key [7]. See section 3.2.6 for a description registries logic. For the purposes of the peg-zone, all that matters is an event is emitted with each change to the registry's state.

The peg-zone is implemented as part of the Kosu network by a special process validators run called a Witness. All validators are required to run full Ethereum nodes, and submit attestation transactions for each of the pertinent events (described above, and in detail below), signed with their validating key pair.

When a super-majority of active validators[13] submit attestation transactions to the Kosu network about a given event from the contract system, that event is "applied" to the network's state by updating the necessary account in the case of a poster, or by applying a modification to the validator set.

[13] Measured in terms of vote power, the number of validators does not matter.

3.4.3 Process description

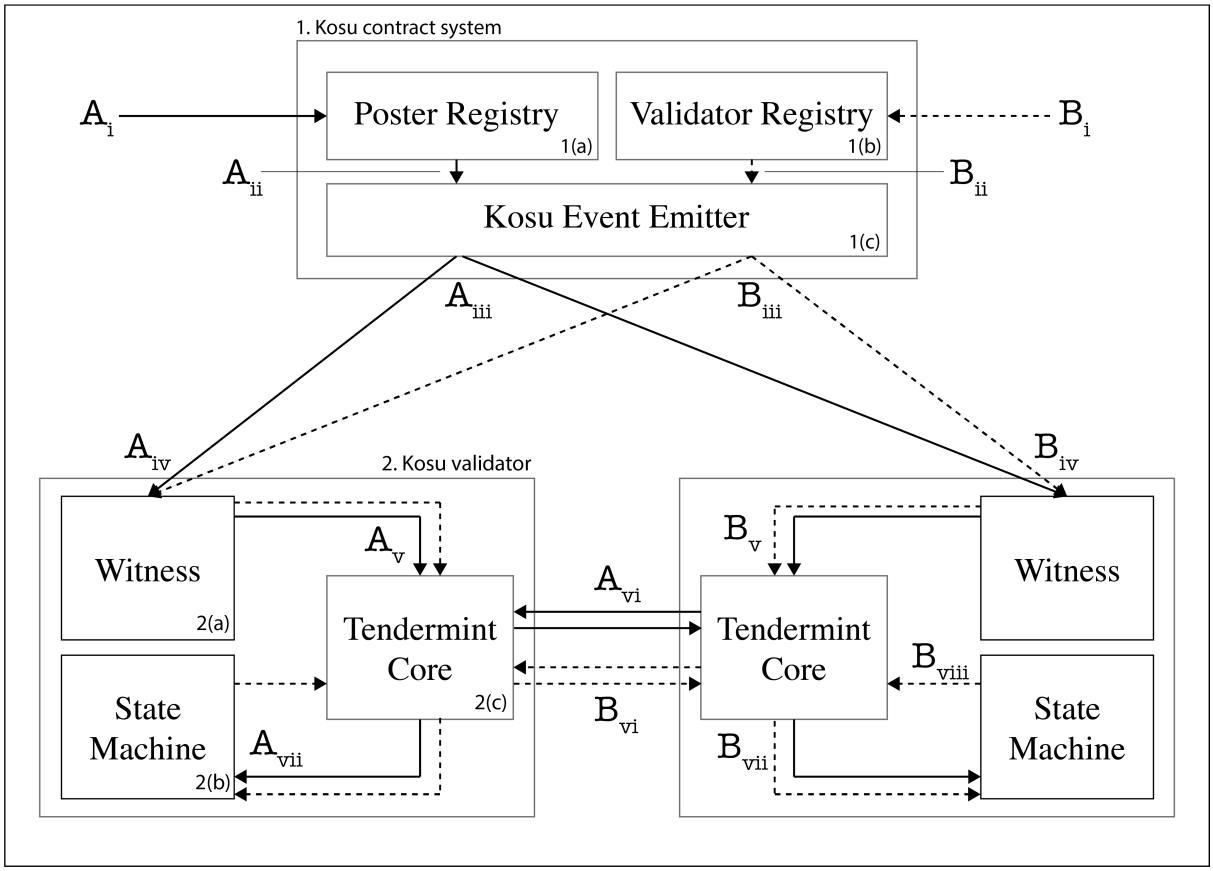

Figure 2: Simplified diagram of the systems involved in the Kosu-Ethereum peg zone. Box (1) shows components of the Kosu contract system, box (2) shows components of Kosu validators. The processes illustrated in red and blue are described below.

Below is a simplified description of the processes involved in updating a poster account (A, solid lines in fig. 2) or modifying the validator set (B, dashed lines in fig. 2) as part of normal peg-zone functionality. The diagram shows only two validators for illustrative purposes, a live network should never operate with less than four validators, the minimum required to tolerate a single fault (byzantine or otherwise).

The process of updating the network's validator set and poster accounts is similar, as both use the same witness infrastructure. However, there are important distinctions between how the events are initially generated, and the transitions they effect within the underlying Kosu state machine.

A) Addition or modification of poster account (shown as A, solid lines in fig. 2):

i) A poster entity submits a bond or unbond transaction to the poster registry.

ii) If the transaction is valid, it is included in an Ethereum block and the registry calls a function on the EventEmitter contract during execution of the transaction.

iii) The EventEmitter contract produces an event log containing the posters address, their new balance, and additional meta-data.

iv) The Witness process on active validators is notified of the event from a local Ethereum node. After a certain confirmation threshold (measured in Ethereum blocks) passes, the Witness will sign a Kosu WitnessTransaction with the validators keypair attesting to the event.

v) The validator's attestation is submitted to their local Tendermint instance.

vi) Tendermint propagates the attestations throughout the network to be validated, executed, and included in a block.

vii) When the transactions reach the underlying state machine, they are added to a confirmation queue where they await sufficient confirmation from a super-majority of validators. Once reached, the state change is applied to the necessary poster account to reflect their updated balance of bonded Kosu tokens.

B) Addition or removal of a Kosu validator (shown as B, dotted lines in fig. 2):

i) A validator listing is confirmed into (or voted out of) the Validator Registry (exact process detailed in section 3.2.6)

ii) During execution of an Ethereum transaction that confirms the validator's listing update, a function call is made to the EventEmitter contract.

iii) The EventEmitter contract produces an event with the validator's Ethereum address, their Tendermint public key, and their staked balance (which is set to 0 in the case of a removal).

iv) A validator's active Witness process receives and parses the event, and waits for the necessary number of Ethereum blocks to be mined above the block the event-trigger transaction was included in.

v) The validator's Witness component signs a Kosu WitnessTransaction attesting to the update, and submits it to the network via Tendermint.

vi) Tendermint propagates the attestation to other validators for inclusion in an upcoming block.

vii) Similar to poster events, new attestations are added to a pending queue until sufficient attestations are received about the event from active validators.

viii) At this point, the process differs from the handling of poster balance updates. When the validator update event is confirmed, existing validators include field in the `EndBlock` response message sent to Tendermint that indicates the adjustment to the validator set. In the case of a new validator, the new public key is assigned a vote weight equal to the number of staked tokens[14]. If the validator is removed, their vote power is set to 0. The update applied at block `H` takes enables a new validator to join the set at block `H+2` [5].

3.5 Incentive Models

3.5.1 Validators

The network described above utilizes a bonded proof-of-stake system to manage sybil tolerance. Validators are required to make a security deposit, also known as a stake, in order to participate in consensus. Tendermint Consensus allows byzantine behavior committed by validators to be easily verified and punished accordingly.

The KOSU network makes a few core assumptions that influence the design of the validator incentive model. Such assumptions are motivated below.

I. Validators are expected to capture both explicit and implicit rewards.

- Central topographical positioning within the network provides validators with unique informational asymmetries that can be exploited for profit.
- Discouraging such behavior without obscuring transactional information is not immediately desirable.

II. Implicit reward capture is highly variable and progressive.

- There exists a direct relationship between network usage and the frequency of exploitable information.

[14] A conversion factor is required to avoid integer overflow of the Tendermint vote power representation.

- Though the value of the informational asymmetries provided to validators is not easily quantifiable, we can expect there exists a positive correlation between network usage and the value of implicit reward capture.

III. The number of validators is negatively correlated with network performance.

- Network security is a function of both the distribution of wealth as well as the total bonded token quantity of validators.
- Comparatively (and in a controlled setting), network performance is purely a function of validator quantity.
- As more validators are added to the network, latency increases absolutely.

These assumptions motivate a market driven fee schedule for a validator set that is dynamic in both size and inclusion. The Validator TCR system (described in section 3.2.6) allows for exactly this. To become a validator, a candidate must bond an unspecified amount tokens (above the parameterizable minimum), which will provide them a proportional influence in consensus, as well as define a requested fee schedule. The expected economic impact of a proposal (via inflation, security, latency) motivates a focal point on a continuum that will eventually resolve in a binary direction.

It is important to note that this validator selection model is considerably more subjective than largest-bond (rank) based proof of stake validator selection mechanisms. With that said, the unique domain requirements as well as the limited impact of malicious validator behavior seem to warrant such subjectivity.

The listing can be challenged at any time by submitting a challenge proposal with a stake equivalent to the stake of the bonded validator. If the challenge succeeds, the validator is removed from the registry and loses their stake, which is distributed to voters on the winning outcome proportionally to token weight with extra reward given to the challenge initiator. This challenge method is a stakeholder-based slashing mechanism that is designed to regulate subjectively malicious value capture behavior.

More explicit byzantine behaviors can be enforced structurally via explicit slashing conditions based on fraud/liveness proofs. Conditions/behaviors that could be discouraged explicitly include double voting, low validator liveness, incorrect event witnessing etc.

3.5.2 Posters

In order to regulate network spam, posters, individuals contributing order data to the network, are also required to bond tokens. Posters are awarded a network access proportional to their bonded token quantity. In this model the inclusion of transactions is not fee based, but rather bandwidth based. This is a significant design feature of the network motivated on the basis that quality liquidity creation should be rewarded, and transaction fees are not compatible in this regard. Individual orders are not necessarily valuable, thus network contributions should be feeless so long as the contributor's incentives are aligned with those of the greater network.

Unlike validators, the behavior of posters is not directly punishable via slashing. Instead, the poster bond is intended to implicitly align the economic incentives of posters with the success of the overall network and create sybil tolerance via token scarcity. As the usage of the network increases, posters contributing quality orders are rewarded via token appreciation. Conversely, there is an economic disincentive for posters to post spam as this would result in a depreciated token value.

In this model, there does exist the potential for "free-riders", particularly speculators benefiting from the appreciation of KOSU value as a result of liquidity providers' work. While not completely ideal, this artifact is not entirely corrupt as the "free-riders" do in fact contribute to the security of the protocol by increasing token scarcity.

3.5.3 Voters

The validator selection model used by the KOSU network is relatively subjective, requiring the active participation of token holders in the curation of a validator set. Due to the magnitude of such decisions, it is important to maximally incentivize participation via explicit participation rewards.

Token holders participating in the curation mechanisms, including those who submit winning challenges and vote correctly in challenges earn KOSU tokens as rewards. This results in an active token distribution towards active and benevolent network actors. It is important to note that all token holders, including those bonded as both validators and stakers, can participate in any election/decision.

4 Token Distribution

4.1 Overview

Token distribution is a key determinant of incentive compatibility for the models defined previously. Kosu's proposed token includes multiple utilities that collectively stimulate wealth distribution to be a relatively dynamic process. To complement such dynamism, Paradigm Labs plans a continuous token distribution via a market-driven, validator defined reward system (outlined previously) and a bonding curve. KOSU's distribution will thus be a function of both a deterministic process, the bonding curve, and an indeterminate process, the validator reward schedule.

Preceding an official network launch, a contract responsible for token distribution will be deployed[15]. The contract will be permissioned to mint additional KOSU tokens based on an algorithmic pricing model. It will allow users to receive KOSU by bonding ETH and vise versa. The pricing model defines a parametric equation that allows curves to be dynamically defined based on the existing token contract state. This construction maintains the correctness of the system by providing an additional independent variable necessary for price adjustment due to token inflation as a result of validator rewards.

4.2 Mechanics

In order to gain intuition about the system and parametric equation used, we will consider the following derivation.

Let us define a slightly exponential family of curves $y = m * x^n$ where y is the price of KOSU denominated in ETH, x is the KOSU supply, n is a predetermined constant within the range $(1, 2)$ and m is the slope which is parameterized upon each bond/unbond transaction. The power function y represents the price of KOSU denominated in ETH and x defines the total KOSU supply. We can thus calculate the total amount paid, which should correspond to the total pool balance, at some token supply j with the following integral calculation.

$$b = \int_0^j m * x^n dx \tag{1}$$

The full integral can be simplified to

$$b = \frac{m}{n+1} j^{n+1} \tag{2}$$

Which can be re-written in terms of m:

$$m = \frac{b(n+1)}{j^{n+1}} \tag{3}$$

Borrowing from Bancor, let us now define a constant called the reserve ratio, r, which gives the quotient of the current pool balance to the product of the token supply, j and the price at this point, p_j [21].

$$r = \frac{b}{p_j * j} \tag{4}$$

Substituting our previously derived representation of b and further expanding p_j we get:

$$r = \frac{\frac{m}{n+1} j^{n+1}}{j * (m * j^n)} = \frac{1}{n+1} \tag{5}$$

[15] To preserve fairness, no tokens will be pre-mined. With that said, the bonding functions will be exponential in order to incentivize early participation.

From which we can calculate:

$$n = \frac{1}{r} - 1 \tag{6}$$

Substituting our newly derived representations of m and n and simplifying, we can derive the following parametric equation.

$$y = \frac{b}{r * j^{\frac{1}{r}}} * x^{\frac{1}{r}-1} \tag{7}$$

Where, in equation 7:

y = KOSU price denominated in Ether
x = quantity of KOSU bonded
j = total KOSU supply
b = total quantity of Ether collateral
$r = \frac{b}{p_j * j}$

From equation 7, we can further derive equations for calculating payouts in both directions. The equation for both payout directions are included below. For a more formal derivation of the equations referenced below, please refer to Bancor's "Formulas for Bancor system" [22].

4.3 Ether to Kosu

$$\beta = j * ((\frac{1+\alpha}{b})^{\frac{b}{p*j}} - 1) \tag{8}$$

α = quantity of Ether deposited

β = quantity of KOSU received

j = KOSU supply

b = total quantity of Ether collateral

p = price of KOSU at j

4.4 Kosu to Ether

$$\alpha = b * ((\frac{1+\beta}{j})^{\frac{p*j}{b}} - 1) \tag{9}$$

β = quantity of KOSU deposited

α = quantity of Ether received

j = KOSU supply

b = total quantity of Ether collateral

p = price of KOSU at j

5 The Matcher Model

There will inevitably exist a wide distribution in the technical sophistication of Kosu users. Traders, for example, may have little understanding of the bonding concept used by the system to maintain access control. Other subtleties, including issues with order collisions and front running, should not be assumed knowledge for general users of the system. As such, second-layer services should provide abstractions to many of the system's complexities. In addition to approving user experience, these abstractions concentrate the network's stakeholders to a more participatory group.

For makers, these services might provide an endpoint with an associated quantity of bonded tokens, allowing users to post limit orders through an GUI/API without needing to bond tokens themselves. These could be structured as free services (entities hold KUSO tokens and expect their liquidity contribution to accrue value via the appreciation of the underlying token) or for a per-post fee.

For servicing takers, these second-layer systems may function similarly to designated market makers in dealers markets; providing direct liquidity with quotes adjusted by a premium for the assumed execution risk.

6 Future Work

6.1 Formalized Governance

A special privilege afforded to the core development team will be the responsibility of managing specific system parameters. These parameters include total bandwidth, the exponential rate of the token bonding curve, finality threshold, etc.

At genesis and for the foreseeable future, the management of these parameters will be at the discretion of the Paradigm Labs team who intend to publicly involve significant stakeholders in an informal collective decision making process.

Paradigm Labs intends to eventually transition such control to a more formal decentralized governance system, following cues from well-established protocol's that are also pursuing similar systems.

6.2 Decentralized Trade Execution

Orders shared on the Kosu network are (generally) public in the sense that any taker can become a counterparty by submitting the signed order message to the layer-one network and settlement contract system specified by the order. With account based layer one networks like Ethereum, these transactions are batched into blocks. In the scenario where multiple parties attempt to fill an order, the order that is included first will settle while the other will fail. This means that the taker whose transaction failed will waste gas and effort.

The 0x project has proposed various solutions to this problem including the introduction of a Trade Execution Coordinator which can enforce execution scenes such as price time priority and selective delay off-chain via specific order construction and a contract extension [24].

The current implementation of the TEC system proposed by 0x is not trustless, but is expected to evolve towards a decentralized solution [25]. This work is particularly important for competitive/liquid markets as it will contribute to reduced spreads by reducing front-running opportunities. Kosu is designed to be agnostic to execution systems and will thus benefit from any progress made at this layer.

6.3 Decentralized Price Oracles

Secure and scalable decentralized price oracles are a juggernaut. Many projects within the decentralized finance space currently require trusted price feeds that provide little accountability for correctness. This is a large hole within the decentralization of many existing protocol architectures.

Recently, the Marble team released Polaris, an on-chain decentralized price oracle for ERC20 tokens. Polaris works by calculating the median of historical checkpoints on Uniswap [26]. Polaris is a step in the correct direction, but will be fundamentally limited in security by the adoption of the underlying exchange primitive, Uniswap. Uniswap provides a compelling automated market making solution for decentralized exchange, but requires active arbitrageurs for price convergence. This means Uniswap's system is effectively price discovery by arbitrage. In our team's opinion, the scalability potential of this solution is limited [27]. As such, we believe the best way to scale a decentralized price oracle is likely off-chain.

There are a few viable approaches to this problem. Perhaps a medianized rate, as determined by a curated set of participants, is sufficient for some applications. However, we believe that a strong decentralized price oracle should enforce price calculation in an explicit and auditable manner. To do so, requires a consistent set of valid orders, as well as guarantees around the public visibility and fillability of such orders. Though complex in construction, Kuso might provide the basis for such a system. Kuso maintains checkpoints to the Ethereum chain and provides strong consistency for a globally maintained order set. Together, these properties could potentially be leveraged to create proofs of validity, visibility and ultimately price.

With that said, the strength of such a system, as referenced in the Uniswap discussion, is heavily dependent on adoption of the underlying protocol. The viability of such a system is also heavily dependent on minutiae regarding implementation. Nevertheless, the possibility of such a system is intriguing.

References

[1] Warren, Will and Bandeali, Amir. "0x: An Open Protocol for Decentralized Exchange on the Ethereum Blockchain."
`https://bit.ly/0x-whitepaper`

[2] Daian, Philip., et al. "Flash Boys 2.0: Frontrunning, Transaction Reordering, and Consensus Instability in Decentralized Exchanges."
`https://bit.ly/flash-boys-paper`

[3] All In Bits Inc., Tendermint, et al. "Tendermint Consensus Reactor."
`https://bit.ly/tendermint-consensus-reactor`

[4] Kunze, Federico, et al. "Two-way Permissionless Peg Zones." Tendermint.
`https://bit.ly/peggy-spec`

[5] Buchman, Ethan., et al. "Application Blockchain Interface Specification (Methods and Types)."
`https://bit.ly/tendermint-abci-spec`

[6] Buchman, Ethan., et al. "Application Blockchain Interface Specification (Applications)."
`https://bit.ly/tendermint-abci-applications`

[7] Buchman, Ethan., et al. "Tendermint Core specification: Encoding and Digests."
`https://bit.ly/tendermint-encoding-spec`

[8] Ramsay, Zack., et al. "Tendermint Core specification: Peer Discovery"
`https://bit.ly/tendermint-node-spec`

[9] Buchman, Ethan., et al. "Merkleized IAVL+ Tree implementation in Go"
`http://bit.ly/tendermint-iavl`

[10] McKinsey Global Institute, "$118 Trillion and Counting: Taking Stock of the World's Capital Markets."
`https://bit.ly/mckinsey-capital-report`

[11] Vogelsteller, Fabian and Buterin, Vitalik (et al.). "ERC-20 Token Standard." Tendermint.
`https://bit.ly/erc-20-specification`

[12] Leslie Lamport, Robert Shostak, Marshall Pease. "The Byzantine Generals Problem."
ACM Transactions on Programming Languages and Systems, Vol. 4, No. 3. July 1982. Pages 382-401.

[13] Tasca, Paola., Tessone, Claudio. J. "Taxonomy of Blockchain Technologies; Principles of Identification and Classification."
`https://bit.ly/taxonomy-of-blockchains`

[14] Goldin, Michael. "Token Curated Registries 1.0." Medium.
`https://bit.ly/token-curated-registries`

[15] Ethan Buchman, et al. "Tendermint: Byzantine Fault Tolerance in the Age of Blockchains."
`https://bit.ly/tendermint-thesis`

[16] Hollander, Nadav. "Dharma: A Generic Protocol for Tokenized Debt Issuance."
`https://bit.ly/dharma-whitepaper`

[17] Paradigm Labs, corp., et al. "Paradigm Foundation GitHub."
`https://bit.ly/paradigm-labs-github`

[18] Paradigm Labs, corp., et al. "Kosu Monorepo: A Monorepo for the Kosu Protocol."
`https://bit.ly/kosu-protocol-repo`

[19] Freyaldenhoven, Nick., et al. "Kosu SubContract SDK."
`https://bit.ly/kosu-settlement-sdk`

[20] Buterin, Vitalik., et al. "Ethereum White Paper."
https://bit.ly/ethereum-white-paper

[21] Hertzog, Eyal., et al. "Bancor Protocol: Continuous Liquidity for Cryptographic Tokens through their Smart Contracts."
https://bit.ly/bancor-white-paper

[22] Rosenfeld, Meni. "Formulas for Bancor system."
https://bit.ly/bancor-formulas

[23] Bloemen, Remco. "0x Roadmap 2019 (part 2): Scalability R&D."
https://bit.ly/0x-scalability-pt-2

[24] Zeitz, Peter. "Research on Selective Delay TEC Process."
https://bit.ly/0x-selective-delay-research

[25] Bandeali, Amir and Berger, Fabio. "0x 2.0.0 Coordinator Specification."
https://bit.ly/0x-coordinator-spec

[26] Pereira, Mykel. "Introducing Polaris: A trustless price oracle for ERC20 tokens on Ethereum Mainnet."
https://bit.ly/polaris-intro

[27] Uniswap. "Uniswap Exchange Protocol Documentation."
https://bit.ly/uniswap-docs