Kosu: A Decentralized Relay Protocol for Smart Contract Based Financial Primitives

Liam Kovatch, Henry Harder
www.kosu.io

April 18, 2019

Abstract

We motivate Kosu: a decentralized network and protocol that encourages the curation of a consistent, high-quality, and 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 rule-set 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 throughput 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. // TODO: review and finalize

Contents

1 Introduction

Blah, blah blah blah. Blah.

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 [2] and a system of Ethereum contracts (that include that protocol's native token).

Kosu is chiefly composed of two interdependent systems that together implement the protocol's core function: to support the decentralized relay of order messages, and to establish a consistent and verifiable set of recent orders. The Kosu contract system supports the protocol's incentive mechanisms, validator curation by stakeholders, and sybil tolerance for the Kosu network, which supports base-layer order book functionality for exchange and liquidity systems built atop it.[1]

The first system is the Kosu blockchain, a collection of globally distributed nodes that collectively maintain and update the network's shared state and "order book"[2], and maintain a one-way Peg with the Ethereum blockchain. 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 comprising the "order book". Validators secure the network by voting on state changes to an Ethereum contract system, processing orders, and reaching consensus with other validators[3] on the canonical chain and the overall application state.

The second and closely related component is the Kosu protocol smart-contract system. These contracts are deployed to Ethereum and implement Kosu's native token and incentive mechanisms, support access control and sybil tolerance for posters, and allow stakeholders to curate the Kosu network's validator set through a novel token-weighted voting system.

Kosu validators run full Ethereum nodes and are responsible for submitting special attestation transactions to the Kosu network about specific state changes to the protocol's contract system. The only contract state changes the network must reflect are updates to users bonded token balances (which affect a order rate limit enforced by the network), and updates to a dynamic registry contract containing the list of validators stakeholders have currently approved.

Posters who wish to leverage the network's decentralized order booking and message relay features can gain write access to the network by bonding any amount of Kosu tokens in a specific 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.

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 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 bandwidth 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 that have been gossiped to them through full nodes, while submitting regular attestations (called witness transactions) to other validators on the network to ensure the state of the Kosu Ethereum contract system is accurately tracked.

The validity of order transactions is based solely 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. If

[1] See section 3 for details on each of these systems.

[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 TODO

[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.

a valid poster signature is included in the incoming Order and the recovered address matches that of a poster account with a non-zero remaining limit for the current rebalance period, validators will accept the order and include it in a block and decrement the posters bandwidth allocation limit for that period. Otherwise, the order is rejected.

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 incredible value to validators in the form of an informational advantage with regards to who sees new orders first.

Validators see proposed and newly committed blocks prior to the rest of the network, and at scale, this information can be monetized. In order for the incentives to remain properly aligned in these conditions, the same aspect of the protocol's design that enables the inflationary reward mechanism, also enables validators to compete by offering to burn tokens for a validating position, which amounts to a payment for the role. The "payment" comes in the form of removing tokens from circulation (burning), which increases the value of all remaining tokens due to the decrease in supply, assuming no or little change in demand.

3 Specification

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

Many nuanced components and processes are referenced at a high level for the sake of brevity and digestibility. Each component of the protocols implementation is open source [10] and should be referred to for the most detailed and up-to-date specification and implementation.

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 allocations, during the network's order verification process.

The order format described below is also designed to allow the "wrapping" of already existing hybrid-decentralized[4] order message formats (such as 0x and Dharma) 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.

[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 on orders regardless of settlement type.

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 [12].

- `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 [9], etc.)
2. Efficient and lossless compression with existing transports and multiple languages in mind
3. Support 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-independent 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 [11] 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 deemed 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 bonded asset validators must stake, and to incentivize the curation of a high-quality validator set (see 3.2.6 and 3.5.2).

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 should be able 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 sub-sections.

- `AuthorizedAddresses` - a registry 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 [4]
[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 [7] 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 Kosu treasury contract

3.2.5 Poster bonding

Posters can...

3.2.6 Validator token-curated registry

The validator set can...

3.2.7 Fraud proofs

The validator set can...

3.3 Tendermint Network

3.3.1 Introduction

The Kosu network, which support

3.3.2 Architecture

Architecture can...

3.3.3 Poster access control

Posters can...2

3.3.4 State model

State can...

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

3.3.5 Transaction types

Transaction can...

3.3.6 Validator curation

Validators can...

3.4 Ethereum Peg Zone

3.4.1 Introduction

Communication between blockchains with differing types of finality guarantees requires the usage of a peg-zone. The Ethereum blockchain currently offers probabalisitic finality for state-modifying transactions included in blocks based on its proof-of-work consensus mechanism. Older blocks are exponentially harder to reorganize [6] 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 [2, 8] engine. As soon as a superiority 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 sub-set 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 [3] 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 design of the system is

3.4.3 Process description

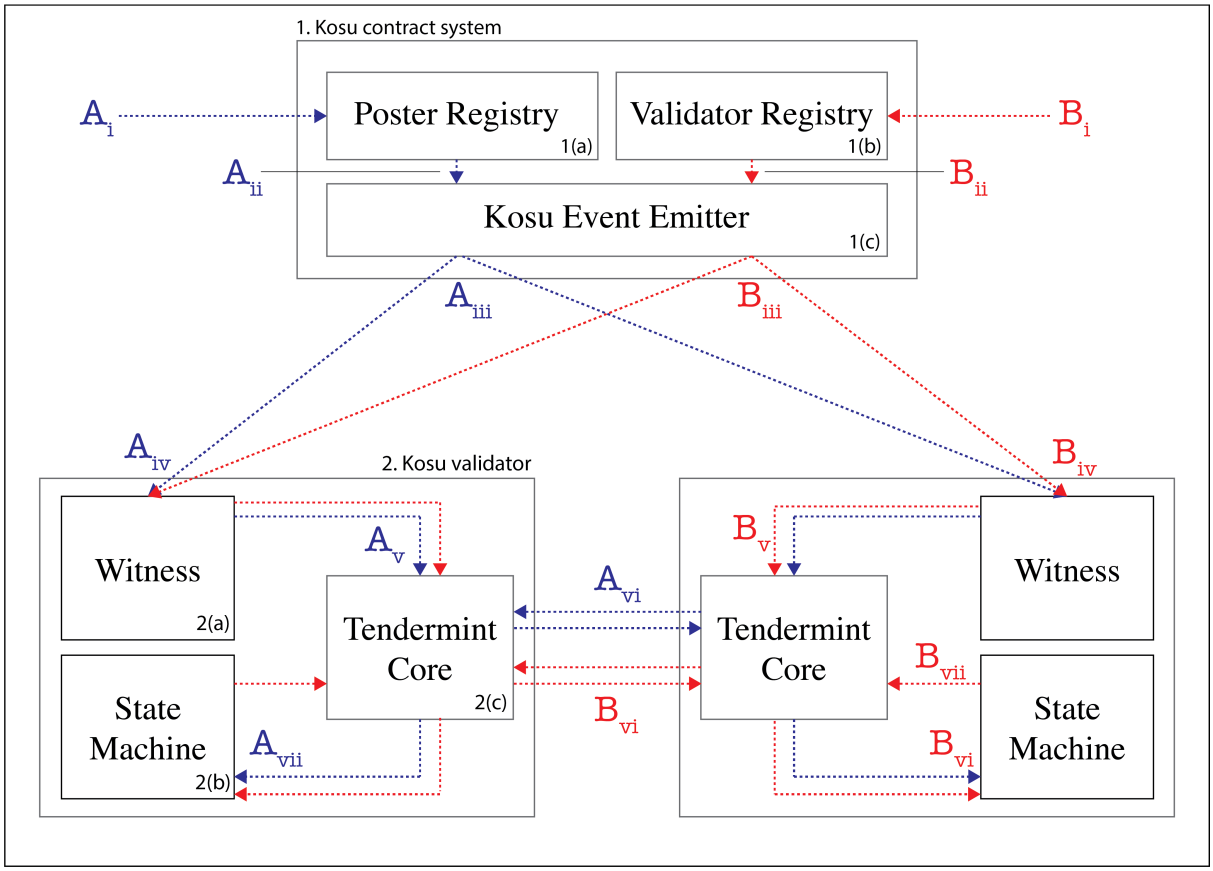

Figure 1: Simplified diagram of the systems involved in the Kosu-Ethereum peg zone. Box (1) shows components of the Kosu contract system, box (2) shows components of Kosu validators. The processes illustrated in red and blue are described below.

3.5 Incentive Models

3.5.1 Sybil tolerance

Poster bonding is the sybil tolerance mechanism used...=

3.5.2 Validator elections

During curation processes, voters can...

3.5.3 Validator rewards

Validators can specify a reward in Kosu tokens... // cover burn case

4 Token Distribution

4.1 Overview

A continuous...

4.2 Mechanics

Such a model...

4.3 Ether to Kosu

ETH to KOSU...

4.4 Kosu to Ether

KOSU to ETH... [5]

5 Future Work

References

[1] Warren, Will and Bandeali, Amir. "0x: An Open Protocol for Decentralized Exchange on the Ethereum Blockchain."
https://github.com/0xProject/whitepaper/blob/master/0x_white_paper.pdf

[2] All In Bits Inc., Tendermint, et al. "Tendermint Consensus Reactor."
https://github.com/tendermint/tendermint/blob/master/docs/spec/reactors/consensus/consensus.md

[3] Kunze, Federico, et al. "Two-way Permissionless Peg Zones." Tendermint.
https://github.com/cosmos/peggy/tree/master/spec

[4] Vogelsteller, Fabian and Buterin, Vitalik (et al.). "ERC-20 Token Standard." Tendermint.
https://github.com/ethereum/EIPs/blob/master/EIPS/eip-20.md

[5] 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.

[6] Tasca, Paola., Tessone, Claudio. J. "Taxonomy of Blockchain Technologies; Principles of Identification and Classification."
https://arxiv.org/pdf/1708.04872.pdf

[7] Goldin, Michael. "Token Curated Registries 1.0." Medium.
https://medium.com/@ilovebagels/token-curated-registries-1-0-61a232f8dac7

[8] Ethan Buchman, et al. "Tendermint: Byzantine Fault Tolerance in the Age of Blockchains."
https://allquantor.at/blockchainbib/pdf/buchman2016tendermint.pdf

[9] Hollander, Nadav. "Dharma: A Generic Protocol for Tokenized Debt Issuance."
https://whitepaper.dharma.io

[10] Paradigm Labs, corp., et al. "Paradigm Foundation GitHub."
https://github.com/ParadigmFoundation/

[11] Paradigm Labs, corp., et al. "Kosu Monorepo: A Monorepo for the Kosu Protocol."
https://github.com/ParadigmFoundation/kosu-monorepo

[12] Freyaldenhoven, Nick., et al. "Kosu SubContract SDK."
https://github.com/ParadigmFoundation/ParadigmContracts/blob/master/sdk/