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

Liam Kovatch, Henry Harder
https://kosu.io

5 June 2018 (Revised April 15, 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 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

Contents

1 Introduction

The... // TODO (coming from google doc)

2 Overview

Kosu is chiefly composed of two interdependent systems that together implement the protocol's core functionality.

The first system is the Kosu network, a collection of globally distributed computers (nodes) that collectively maintain and update the network's shared state. The network supports basic order booking functionality, access control for user accounts, and cryptographic proofs of the existence of state contents. Validators secure the network by processing transactions, creating blocks, and reaching consensus with other validators on the canonical chain and 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 mechanism, support access control and sybil tolerance for posters, and allow stakeholders to curate the Kosu network's validator set.

Kosu validators run full Ethereum nodes and are responsible for submitting special attestation transactions to specific state changes to the Kosu contract system. The Kosu networks knowledge about the state of the contract system is limited to only what is strictly necessary, such as maintaining a registry of users allowed to post orders to the network (posters).

Posters who wish to leverage the network's decentralized order booking 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, validators update the networks 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 account-limit mapping based on in-state 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 continuous process of balance updates on the Ethereum Poster Registry as a result of bonding/unbonding by posters to be mapped to 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 concurrently 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.

Order transactions are verified purely based on the signature of the poster that submitted the order to the network, and some basic data structure requirements that enable signature recovery. If a valid 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.

... DESCRIBE ORDER BOOK AND QUERY MODEL, PROOFS ETC

3 Specification

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 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.

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. // link to definition of this term

All current SubContract implementations are in Solidity and designed for use with Ethereum, however, any language and blockchain that supports contract-like settlement implementations that satisfy the interface above, and the correct signature scheme, can be used with Kosu.

Method name	**Returns**	**Params.**	**Description**
`makerArguments`	`string`	-	Returns a JSON string containing all the fields required for maker orders.
`takerArguments`	`string`	-	Returns a JSON string containing all the fields 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.

3.1.2 Order message format

A simple order format is defined based on the SubContract interface, and the requirements for signature verification... // TODO

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: Generically describes the data structure that represents an order message.

3.2 Ethereum Contract System

3.2.1 Introduction

The Kosu Ethereum contract system...

3.2.2 Permission model

Access control within...

3.2.3 Architecture

The contract system uses a modular...

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...

3.3 Tendermint Network

3.3.1 Introduction

Tendermint cadddn...

3.3.2 Architecture

Architecture can...

3.3.3 Ethereum peg-zone

Ethereum can...

3.3.4 Poster access control

Posters can...2

3.3.5 State model

State can...

3.3.6 Transaction types

Transaction can...

3.3.7 Validator curation

Validators can...

3.4 Incentive Models