Simplified Smart Contract Environments

A more efficient solution for coordinating decentralized web-services and creating large-scale data consensus, provenance, and confirmation by using storage instead of blockchains.

Alexander Morris Kiefer Joe Burgess Abel Sebhatu

Abstract

In this paper, we outline how smart contract execution can be achieved using a network of consensus nodes and a decentralized storage container, without the direct need for a proof of work blockchain.

In Gradual Consensus, as developed in the Koii Network, Consensus Nodes perform reliable compute operations and report their results back to a group of high-reputation Bundler Nodes who batch high volumes (theoretically millions) of transactions at a time.

This technology was originally developed to support decentralized web-scraping, followed by attention tracking in the Proofs of Real Traffic game. We have now expanded the model to support a wider set of applications, each with its own particular consensus rules, but running on a common set of nodes.

In each case, the designer constructs three functions (submission, rewards, and audit), an executable bundle, and a state object, and deploys them to a storage medium to begin the game. Then, any nodes who stake enough tokens will be able to join and fulfill the task.

The Koii Network implements this structure natively as Koii Tasks, and provides out-of-the-box consensus across Koii Nodes and Applications.

Feedback is welcome, so please contact hello@koii.network.

Contents

1. Motivation

While many decentralized smart contract networks have evolved in the wake of Ethereum's success, most have focused on building Turing-complete machines based on Proof-of-Work. Generally, even if these are built using a Proof-of-Stake model, they are designed for transaction finality instead of development speed and flexibility. *For many applications, this a nonoptimal design choice and it limits utility significantly by excluding all non-deterministic outcomes.*

2. Gradual Confirmation Game Overview

A gradual confirmation consensus mechanism employs Proof-of-Stake incentives combined with large scale storage to provide a framework for non-deterministic and lazilly executed smart contracts, while also making development significantly easier. This work is based on the private-blockchain frameworks developed by Hyperledger and other projects, and combines a RAFT Consensus model [6] with an incentivised audit process to confirm large amounts of data more slowly.

Whenever new data is proposed, a Gradual Confirmation Vote is triggered, in which participating nodes will automatically join and provide verifications. Vote data is collected by 'Bundler' nodes, which aggregate the consensus outcome and then submit it on-chain to update the state. 'Bundler' nodes are held accountable through a combination of audit functions and stake-based incentives.

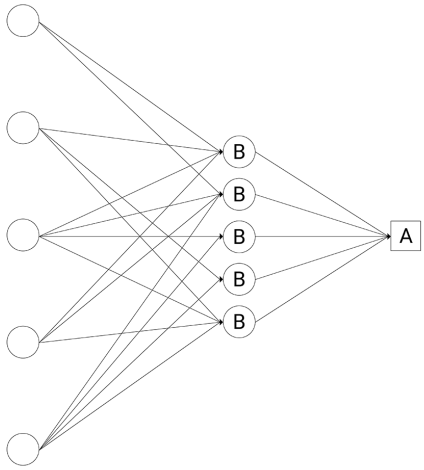
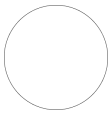


Figure 1: *Bundler Nodes ('B') gather vote and payload data from general nodes and write it to the on-chain archive ('A') as permanent state updates, ultimately performing optimistic roll-ups with integrated auditing and incentives.*

2.1 Participants



Nodes
Standard Nodes perform computation tasks and either write payloads or verify existing payloads by voting.

Bundlers
Bundler Nodes gather vote data and payload data from all other nodes. They are **responsible for writing on-chain updates.**

Typically, bundlers are expected to stake much more because their voting power is scaled to the amount staked and the number of votes they submit.

Archival Storage
A storage network is used to permanently archive consensus outputs, and is used as the foundation for audit votes.

In the default Koii implementation, this role is filled by the Arweave Permaweb.

2.2 Voting Power

Gradual Confirmation ensures reliable results by scaling the weight of each node based on the amount of tokens staked in the task, the length of time their stake is locked, and their reputation such that Voting Power (Pv, below) is represented as:

$$Pv = Kstaked * Tlock$$

When a node passes their votes to a bundler, the bundler calculates the voting power. When Bundlers perform on-chain state updates, they declare the total voting power of their constituents, and publicly display proofs during the audit period. Rewards are distributed to bundlers based on their voting power as a portion of the total, ensuring that the bundlers will attempt to gather as many votes as possible before updating the on-chain state.

2.3 Voting Rounds

A voting round has four discrete steps, which together take anywhere from 6 to 48 hours, and can be delayed or extended if audits occur.

I. **Task Execution & Vote Submissions**
During the first stage of a voting round, non-bundler nodes compete to run a ‘Task Bundle’ (executable) locally and generate payloads. Once a payload has been generated per the requirements of the task, the node will submit it to one or more bundler nodes, who track the order of submissions and tally votes for different payloads.

A. Votes are received as signed JSON Web Tokens (JWTs), and are addressed to each bundler directly (meaning that they are unique and one bundler cannot copy JWTs from another bundler and claim them as their own)

B. Bundlers must provide receipts whenever they are given a ‘vote’ JWT, which a voting node will hold until their content is added to the chain.

II. **Bundler On-Chain Voting**
Once a sufficient number of votes have been received (as defined by a ‘Distribution Function’) the Bundlers will trigger an on-chain state update, and self-declare their ‘Voting Power’ based on all votes received.

* The first Bundler to declare a distribution writes it on-chain, ensuring it is sequenced, and this triggers a veto vote by other Bundlers, who compare their results to the first Bundler’s and automatically decide whether to confirm or deny the vote.

III. **Audit Period**
Once the votes have been submitted, the task enters a review period, during which the Bundlers broadcast all of the JWTs they received, as well as the ‘Distribution’ that they have voted for on-chain. It is here that any node can vote to trigger an audit.

IV. **Confirmation**
If an audit is not triggered during the Audit Period, then the task is considered complete, and the rewards will be paid out after a predefined delay period.

2.4 Rewards Distribution and Timing

Depending on the application, it may be necessary to delay the rewards distribution event significantly. This further incentivizes nodes to behave honestly since the cost of misbehavior increases over time without being able to verify the success of an attack strategy until after a full audit period has elapsed.

2.5 Audit Rewards and Incentives

When an audit is triggered, the votes are submitted to non-implicated[1] bundler nodes, and sequenced to establish the leader-follower order. The first node to vote receives the largest share of the rewards, and subsequent followers receive a smaller share the later they join the vote. This incentive structure encourages nodes to audit more frequently, striving to receive the most rewards.

3. Attack Vectors and Prevention

While not all malicious actions can be predicted, most are fairly obvious due to the incentive structures at play. Of these, we have worked to establish clear solutions and penalties as necessary.

3.1 DDOS via Audit Abuse

Because audits freeze the state update process, successive audits of the same task (or it's audit votes) has the potential to cost a significant amount of network power while preventing resolution of the game. Two factors make this vulnerability difficult to exploit: first, an audit attack would still require a majority vote, which is akin to a 51% attack on the task level, and second, we propose to implement a cost for auditing, and plan to increase it each time a new audit is triggered on the same task.

3.2 Vote Suppression by Bundlers

By scaling the voting power (and thus rewards) of the bundlers to the number of votes they collect and bundle, we push them to try to acquire as many votes as possible. As a result, bundlers who exclude certain voters will receive not only less rewards, but also less input or power over the state update outcome.

[1] A 'non-implicated' bundler is any node registered for that task which did not participate in the last vote.

3.4 Stake Flipping

To prevent network participants from flipping between one task and another, and thus destabilizing the voting process, we implement a variable burn cost on Stake Withdrawal, proportional to the time of the lockup remaining. This enables all nodes to withdraw from stakes at any time, but forces them to pay a fee proportional to the voting power they have already used from that stake lockup.

3.5 Inconsistent Node Behaviour

In the event that a node submits multiple different votes to different bundlers, these votes will be displayed during the audit period, and since they are signed already and expected to be unique to a specific voting round, they can be used as the basis of an audit to slash the voter's stake.

4. Assumptions

While the Gradual Confirmation approach can likely function without issues in an adversarial environment, we elect to test it within the controlled space of the Koii Network. Pre-sale and gradual release of tokens from the Koii ecosystem have resulted in a list of verified partners who will help to launch the network with a reliable federation, until further decentralization can be achieved. As the network grows, this will set a standard of confidence, ensuring future participants are held to the same level of accountability.

4.1 Honest Initial State

Since the majority of nodes in a Proof of Stake network represent investors, it is reasonable to assume that they will behave reliably for at least the initial period of the network, during which the reputation of each node can be established through verifiable interactions.

4.2 Honest Node Operation

The Koii Network provides open-source tools for running a node across platforms, so we expect that a majority of nodes will run the correct node client. This makes it possible to distinguish bad actors based on the similarities of their deviances from the core community, allowing the reputation heuristic to grow in value over time as wayward nodes are penalized.

4.3 Reasonable Rewards

The purpose of this network is to establish channels to create value by gathering information, and so we assume that the rewards for a particular task will be significantly lower than the stake required to interact with the tasks.

4.4 Lockup Horizon

Since Stake Lockups cannot be withdrawn without paying a burn fee, we predict that most nodes will not make stakes that are longer than their investment horizon (i.e. 100 years) since this would result in a significant burn of their tokens, and ultimately devalue their investment. As a result, we expect that a market equilibrium will be established at a stake lockup time which is proportionate to the current confidence of investors in the network.

4.5 Availability of Secondary Reputation Data

Projects like LeyLine already offer NFT-based reputation systems for tracking non-profit volunteer work and participation. Through partnerships with these systems, a DID system can reasonably be expected to evolve to identify nodes who are trustworthy participants in tasks.

5. Koii Tasks and Generalized Implementation

A Koii Task generalizes and incentivizes the implementation of Gradual Confirmation, which asks Koii Nodes and Bundlers to join a game. The common standardized structure makes it possible to deploy consensus aggregation games with flexible reward and audit mechanisms, which write permanently and inexpensively to storage networks like the Arweave Permaweb.

5.1 Task Structure

A standard Koii Task has three main components:

I. **Payload Format**
This defines a common format for results which will be submitted to bundler nodes. This is implemented as the *submitPayload* method.

II. **Distribution Rules**
This defines how rewards can be earned, and how they are divided between participants. This is implemented as the *distributeRewards* method.

III. **Audit Rules**
This defines how rewards can be earned, and how they are divided between participants. This is implemented as the *audit* method.

5.2 Development

Koii tasks can currently be written in JavaScript, TypeScript, or other WASM compatible builds, but can generally fall into one of three classes:

A. **External Data** (deterministic)
These tasks parse non-archive data and allow Nodes to submit specifically structured payloads as Atomic Data (NFT-ified), then vote on the validity of each other's results.

In these cases, the top payload receives the bulk of rewards, with a small amount being given to other payloads that receive statistically significant votes.

Examples: Web scraping AKA GET, Store, Catalogue

Use Cases: Oracle Services, Content Aggregation, or other data ingestion

B. **External Data** (non-deterministic)
Some tasks may also seek to record subjective information, and can perform a similar operation to (A) without the need for a 'winner.'

Examples: Surveys, Image Tagging, Bias, or Fact Checking

Use Cases: Indexing large data sets, developing AI training data, or enriching existing Atomic Content Archives.

C. Services Actions
In some situations, such as with incentivised caching, there is no need to submit a payload, but the goal is instead to monitor nodes for providing access to a certain set of data, or remaining online in a certain capacity.

In these scenarios, it may be possible to skip the payload submission process, and instead simply distribute rewards evenly as long as there's been no failed audits. (i.e. no ongoing game, but audit nodes anyway in case they turn out not to be doing as they are supposed to)

D. Verifiable Actions
Some other applications might include things like message-passing or validating smart contract executions (i.e. Kyve validators) but will require further exploration.

5.1 Deployment

When a Koii Task is deployed, the creator writes an executable file, which they upload to the Archive and then link to their Koii Task contract. Typically contracts can be extended from common templates like the ones in the previous section, and should not require a significant amount of configuration.

5.1.1 Bounties

When deploying a task the creator first locks bounty tokens into the contract, ensuring that a reward is available. Withdrawing from a bounty is not currently possible, but would likely require a waiting period as well as a scaling burn rate.

Once submissions have been made for the task, the bundlers can call the ‘distributeRewards’ function and pass in the payloads they have collected along with a proposed distribution. This action triggers a voting round and releases rewards if it is confirmed.

5.1.2 Beneficiaries

In the event that NFT-ified Atomic data or content are created by a task, the creator may be eligible for KOII attention rewards, and should define a proper scheme to split those among themselves and participating nodes, if necessary. Generally, this can be established as a part of the executable bundle, which will include the NFT’s initial state (or format thereof) and defines the ownership breakdown.

5.1.3 Audit Fallback Mechanism

When an audit is successful, the audited party will have their voting power reduced to zero. In the event that this is a Bundler Node, this means that the runner-up will be considered to be the winning party, and thus will be subject to a second audit period, during which they must produce valid proofs or lose their stake as well. If no runner-up exists, then the network will simply forget that the task occurred, and re-attempt the task in a new voting round.

References

This work is based heavily on and influenced by the existing literature on decentralized blockchain networks, in particular Bitcoin, Ethereum, and more recent projects like Polkadot and Avalanche, and would not be possible without the help and support of Sam Williams, the creator of Arweave, and Taylor Gerring of the Ethereum Foundation.

1. Ethereum Yellow Paper
2. Avalanche Consensus Paper
3. Polkadot Whitepaper
4. Arweave Yellow Paper
5. J. Adler, M. Quintyne-Collins, “Building Scalable Decentralized Payment Systems”, Link
6. Hyperledger Architecture Documentation
7. Mimble Wimble