# Engineering Design and Incentive Specification for Aion-Unity

Ali Sharif
ali@aion.network

August 29, 2019

*"Most magic numbers in blockchain protocols are determined through careful analysis."*
— @VitalikButerin, *Twitter*

**Abstract**

This document is a complete specification for the Aion-Unity upgrade, documenting the design of the staking, delegation and incentive schemes introduced in the release.



## Acknowledgements

This specification was produced in collaboration with the entire R&D staff at the Aion Foundation, with major contributions from Yulong Wu, Yao Sun, Dipesh Pradhan, Jeff Disher, Aayush Rajasekaran, Shidokht Hejazi, Sam Pajot-Phipps, Yunfei Zha and Ge Zhong.

# Contents

# 1 Introduction

This document aims to specify all subsystems required for the implementation of the Unity consensus algorithm [WZS19] as an upgrade to the Aion blockchain.

The key words *MUST*, *MUST NOT*, *REQUIRED*, *SHALL*, *SHALL NOT*, *SHOULD*, *SHOULD NOT*, *RECOMMENDED*, *MAY*, and *OPTIONAL* in this document are to be interpreted as described in RFC 2119 [Bra97].

# 2 Staking and Stake Delegation Protocols

The Unity consensus algorithm combines both proof of work (PoW) and proof of stake (PoS) [WZS19]. This section of the specification outlines the implementation of the PoS subsystem in isolation, to simplify exposition. PoS requires system participants to *stake* their Aion network tokens; we specify the mechanisms to perform *staking*, and the implementation requirements on the core blockchain protocol & ancillary user interfaces required to interact with the system.

## 2.1 Fixing Terminology

In the PoS subsystem of Unity [WZS19], any coin holder of a system can run a full node in the blockchain network in order to produce PoS blocks; we refer to these coin holders as *stakers*.

In order to support a staker's participation in PoS consensus, coins must be rendered immobile (i.e. they cannot be freely transferred to other users) via their submission into the staking system; these coins are referred to as ***stake***. As long as these coins are held (on behalf of the owner) in the staking system, they contribute to the total influence of the staker in the PoS consensus (as discussed in the Unity paper [WZS19]).

When a coin holder casts their coin as stake to a staker, we call this action a ***vote*** or ***bond***. Notice that we to use the word "stake" strictly as a noun throughout this document and the implementation; this is done to avoid confusion with the action of casting coins to a staker, for which we use the verbs *to bond* and *to vote* interchangeably.

The coin holder can either vote for themselves (increasing their power as a staker in PoS) or they can vote for another staker. The action of voting for another staker is referred to as ***delegation***. We further discuss the motivation for delegation in §2.1.1.

Any coin holder that votes for another staker (in order to earn a share of PoS block rewards in exchange for increasing the power of said staker in PoS) is referred to as a *delegator*. Note that a delegator is not a staker, since they don't operate a node and contribute to PoS security.

At any point, a delegator or a voter can choose to withdraw their bonded coins (reducing the PoS power of the staker delegated or voted to). When a delegator withdraws their delegated coins, they are said to ***undelegate***. When a staker withdraws their voted coins, they are said to ***unvote*** or ***unbond***. In either scenario, when the coins in question get unbonded in the staking system, they are subject to a post-lock period ([WZS19] §4.1.1). After the post-lock period has elapsed, their coins exit the staking system and become ***liquid*** (free to transfer to other users).

A delegator is allowed to ***transfer*** their delegation between stakers (subject to stipulations discussed in a subsequent section).

### 2.1.1 The Need for Delegation

In the PoW paradigm, mining pools allow participants with modest hash power to *pool* their resources with strangers to achieve a more reliable outlay of rewards. Similarly, in PoS networks, *delegation* allows users to transfer their rights to participate in the proof of stake (PoS) protocol to *stake pools*.

The rationale for stake-delegation is very much in-line with the rationale for hash-power-pooling: since PoW miners controlling small amounts of hash power are not expected to run a full node in order to write blocks on rare occasions, we should not expect stakers with a small amount of stake to do so either. Furthermore, while miners are generally technically proficient and commit their time to maintaining on-premises infrastructure, owners of stake in the network might lack the expertise or time to do so. Even if one has the willingness to operate a staking node, one might have too little stake to cover operational costs.

Delegation allows all coin holders of Aion to contribute to PoS security, regardless of the amount of coins they own, or their technical capabilities.

## 2.2 Solution Synopsis

### 2.2.1 Staking Contract

All logic related to staking will be implemented on an AVM smart contract called the *Staker Registry contract*. In order to participate in staking, coin-holders must send Aion to this contract, where the balance will reside, until the user decides to *unstake*. Upon unbonding, after a post-active period has elapsed, the balance will be returned by the staking contract to the user's address (see [WZS19] §4.1.1 and §5 for rationale and further exposition). Note, that the pre-active period as defined in [WZS19] §4.1.1 will be zero (i.e. no pre-active stake locking).

This contract is core to the system; correct functioning of the protocol (PoS) depends on the state of this contract, in-so-far-as the lifecycle of all bonded coins are managed by this contract.

### 2.2.2 Delegation Protocol

There exist a continuum of strategies to realize delegation in Aion, which can be characterized by the following extrema:

- **Open**: No delegation features implemented in the core protocol (e.g. PoW, where hash-power-pooling is an off-protocol activity). In order to delegate stake, custodial staking pools would define their own payout schemes, user interfaces, fee-structures, etc.
- **Opinionated**: The core protocol defines a rigid stake delegation protocol. Examples of such protocols include Cardano-Shelley [KBC19] and Tezos [Goo14]. This protocol would clearly define all stake delegation-related interactions between stakeholders and pools, like reward distribution schedule and fee structure.

We take an approach that lies in-between the two extremes. In our approach, the delegation system is completely decoupled from the core protocol. The staking contract exposes a very simple interface to enable stake delegation (while maintaining security-related system invariants). This enables the development of arbitrary delegation protocols through the use of AVM smart contracts we will refer to as *delegation contracts*; these contracts will primarily be responsible for distribution of rewards to delegators and managing the stake life-cycle, while it's delegated to a pool. Figure 1

offers a component-level sketch of how the purported delegation contracts relate to the core Staker Registry contract.

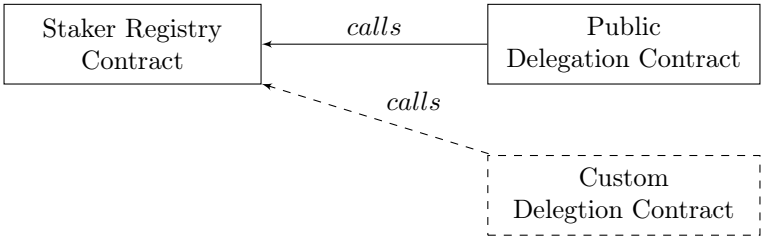


Figure 1: Contracts implementing staking and delegation in Aion-Unity

**2.2.2.1 Public Delegation Protocol** A key goal in Unity system-design is the engagement of all Aion coin-holders in staking. To serve that goal, we further specify an opinionated *public delegation protocol* (implemented as a delegation contract) we call the **Aion Public Delegation Standard (ADS)**. The corresponding contract is called the *Pool Registry*. This Aion-Unity upgrade will be shipped with the ADS and all foundation-supported staking user interfaces will support delegation as specified by this standard.

Furthermore, a clear specification and reference implementation for a public delegation standard allows third party tools like wallets to rapidly implement Aion stake-delegation in their products, to further amplify the accessibility of participating in staking as an Aion coin holder.

**2.2.2.2 Custom Delegation Protocols** Since the public delegation protocol (ADS) is just another contract, leveraging the API exposed by the Staker Registry contract, one can theoretically construct any on-chain stake delegation protocol, build tools to support it, etc.

### 2.2.3 Other Stake Interactions

In addition to enabling the construction of stake delegation contracts, the Staker Registry enables a few obvious (and non-obvious) stake-related interactions:

**2.2.3.1 Private Staking** For technically inclined users interested in running the equivalent of PoW's *solo-miner*, they can simply sign up to be stakers in the Staker Registry, bond their coins to themselves and run an Aion-Unity node to produce PoS blocks when "selected" by the algorithm.

A very simple CLI will be shipped as part of the Aion-Unity upgrade to easily enable private staking capabilities.

**2.2.3.2 Direct Delegation** The design of the Staker Registry enables any user with coins to *vote* for any active staker in the Staker Registry which enables:

- ***Off-chain Stake Pools***: Users who do not wish to use the on-chain delegation functionality, can come to agreements off-chain about fees and rewards splitting with some registered staker, and bond their stake directly to them in the Staker Registry. When the bonded staker gets compensated for PoS participation via block rewards, they would presumably split the rewards with the users (off-chain), based on the staking power contributed to the staker by the user.

- ***Altruistic Delegation***: A user can (inadvertently or deliberately) bond their coins to some staker registered in the Staker Registry, with whom they have no out-of-band agreement on rewards splitting. In such scenarios (since there is no built-in mechanism for rewards splitting in the Staker Registry), the staker in question is not obligated to share any marginal block rewards earned by an increased PoS power vis-à-vis the "altruistically" delegated tokens. Note that at no point is a user at risk of loosing their coins, only loss of PoS rewards which one could have received if one staked via private staking or delegation.

#### 2.2.4 Stake-Delegation User Interfaces

Very specific guidance and requirements are laid out in this document with respect to the responsibilities of any user interface (UI) for the Aion Public Delegation Standard (ADS). The specifications are requirements for what the UI must achieve, but no guidance is provided as to how to achieve it for implementation flexibility based on the medium (desktop, web, mobile), target audience, etc.

### 2.3 System-Level Requirements

The following are key functional, non-functional and security requirements for the Staker Registry and Pool Registry (ADS) contracts in Aion-Unity:

#### 2.3.1 Staker Registry Contract

- **Pseudo non-custodialness**: Technically, the term non-custodial in the context of PoS refers to systems where the coins never leave the user's account while the user is participating in staking, letting the user move those coins without restriction. Due to the design of Unity consensus [WZS19], restrictions on a staker's coins must be imposed (such as immobility and post-active period) to satisfy consensus security requirements. To reduce implementation complexity [YZW19], the user must send their coins to the staking contract. This requirement states that any coins sent to the staking contract cannot be withdrawn by anyone other than the staker (i.e. the coins cannot be withdrawn by any attacker looking to abuse the contract).
- **Post-active period enforcement**: There must exist no code-path to unstake coins without the coins going through a post-active period.
- **Sovereignty over stake**: In delegating coins, the delegate must not be able to affect the stake in any way (e.g. unbond, transfer delegation, etc.)
- **Auditability of delegation**: All stake delegations must be publically visible on-chain.
- **Low-penalty delegation transfers**: Users should be able to transfer stake from one staker to another via an on-chain transaction, with minimal time-lock costs.
- **Mitigate key exposure**: The node run by staking pools will need to have some key that is used to sign produced blocks. In case of an incident where the node is compromised, it should be possible for the stake pool operator to revoke this key and replace it with a new one. This should not require any action from the delegator.
- **Space and time complexity**: All new rules must be computable within *reasonable* space and time complexity.

- **Minimize economic attacks**: An economic attack arises where the costs incurred by the operators are not covered by the fees on the users of the system. Such situations allow users to impose costs on operators without paying the full costs themselves.

#### 2.3.2 Pool Registry

- **Sybil attack protection at the staking pool level**: An adversary can take over the network by registering a large number of stake pools, hoping to accumulate enough stake to mount an attack just by people randomly delegating to them. This attack should be made unfeasible by requiring stake pool operators to allocate a finite resource to each individual pool they register; this cannot be the cost of running a node.
- **Handle inactive stake pools**: Stake pools can cease to operate (lost keys, interest, etc.); we want to minimize the effect of this to the security and liveness of the system.
- **Space and time complexity**: All new rules must be computable within *reasonable* space and time complexity.

### 2.4 Coin Lifecycle in Staker Registry

Figure 2 outlines the *states* that a coin be in, as it makes it's way through the bonding lifecycle in the Staker Registry.

- ***Liquid***: A coin is liquid when it is owned by the user's account. In this state, the user is free to transact in the system (e.g. transfer to another user, stake, pay transaction fee for smart contract interactions, etc.).
- ***Bonded***: A coin is bonded if is has been cast as a "vote" (§2.14 to a staker in the Staker Registry.
- ***Transfer-pending***: Stake can be transferred from one staker to another in the Staker Registry. When this action occurs, the coin must be locked for some short period (transfer-pending period) in order to not violate security invariants of the Unity consensus. The transfer lockout period is required to defend against a version of the state grinding attack (as described in [WZS19]):

  > *Consider a staker who has registered N stake pools. If the transfer of stake was instantaneous, then for all N pools, the staker could compute PoS "time-to-next-block" with various allocations of stake across the staker's N pools, and select the allocation that maximizes their probability of winning the next block (assuming they win the current block and can include these transfer transactions in the current block).*

- ***Thawing***: Any time a coin has been staked (either through the staking contract or the delegation interface), a thawing period must be applied to preserve Unity consensus security invariants [WZS19]. After the thawing period (defined in some number of blocks elapsed since the unstaking action) has elapsed, the coins go into the liquid state (i.e. get returned to the user's account).

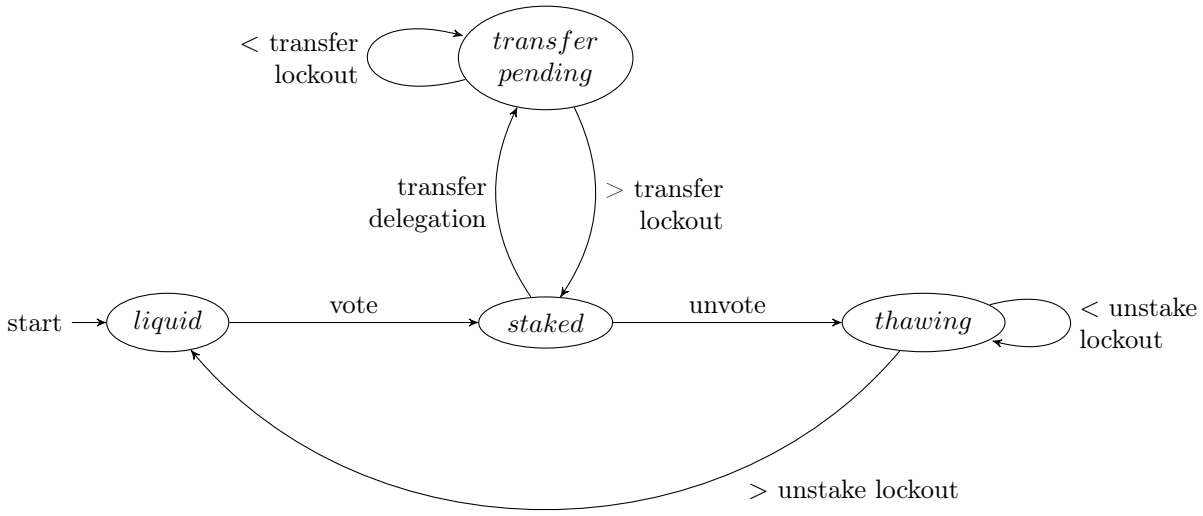


Figure 2: States that a coin can be in within the skating and ADS system

## 2.5 Delegator Actions

The following are the actions that an Aion coin holder, seeking to delegate staking rights, can perform with respect to the Pool Registry contract:

- **Stake Management**
  - ***Delegate stake***: When the user delegates to a pool, they invoke a function in the staking contract, with the cold address of the staking pool as a transaction parameter. In addition, they need to select whether or not they want to enable the *auto rewards delegation* feature, which will automatically move their stake into the delegated state (i.e. stake will not accumulate in the rewards-pending state, and any withdrawals will be subject to an unbonding period). As part of this transaction, the user will send the amount of coin they would like to delegate-stake. Note that there is a minimum amount requirement (in number of coins) for delegations (see §2.10 for rationale).
  - ***Undelegate stake***: When a user decides that they want to withdraw any fraction of their funds from the staking service, they can perform an unstake (or more specifically un-delegate) action in the staking contract. For a period of time measured in number of blocks since the unstaking action, the coin will be in the thawing state; it will be held in the staking contract, but will neither contribute to stake securing the system, nor will it be liquid until the unbonding period has elapsed (see [WZS19] for details).
  - ***Transfer-delegation***: The user should be able to, without triggering the unbonding period, transfer the delegation of any proportion of their stake to another staking pool.

- **Auto rewards delegation**:
  - ***Opt-in auto rewards delegation***: If the user did not opt into the auto-rewards delegation scheme, they can do so at any time while their stake is delegated, by sending a transaction to the delegation contract.
  - ***Opt-out auto rewards delegation***: If the user chooses to opt-out of the auto-rewards delegation scheme, they can do so at any time while their stake is delegated, by sending a transaction to the delegation contract.
- **Rewards management**:
  - ***View rewards***: The user should be able to publicly view all accumulated rewards, including all rewards distribution events, given any delegator-address.
  - ***Withdraw rewards***: The user can choose to withdraw accumulated rewards to their address. No unbonding period is applied to this amount; as-soon-as the withdraw transfer is committed on-chain, the user should be able to see the liquid balance in their account.
  - ***Re-delegate rewards***: The user can manually re-delegate their stake, to increase the total amount of bonded stake they have committed in the system.

## 2.6 Pool Operator Actions

The following actions can be performed by the staking pool operator during the course of the life-cycle of the staking pool:

- ***Register***: To facilitate easy discovery of stake pool by users of the system, as part of pool initialization, every pool must register with the Pool Registry contract. This registry will contain a list of all active pools. The registering pool must provide the following data:
  - *Fee charged*: A number between 0%-100% (up to two (2) decimal place precision), which indicates the fees the pool charges for provided service. See §2.14 for details on how this number is applied to block rewards and fees are distributed to pool operators.
  - *Metadata URL*: The pool operator must host a JSON file at this URL (HTTPS over TLS), containing the metadata that is displayed in ADS user interfaces (§2.13). See the section on the metadata protcol (§2.8) for requirements on what this JSON object should contain and other ancillary concerns.
  - *Metadata content-hash*: This is the Blake2b hash of the JSON object hosted at the metadata URL. This is used as an on-chain commitment of the data hosted at the metadata URL. See §2.8 for details on the function of this content-hash within the metadata protocol.
  - *Coinbase address*: This is the public key that receives any fees collected by the pool operator (see §2.15 on the key-management scheme in the ADS).
  - *(Block) Signing address*: This is the address corresponding to the secret key that the pool operator will use to sign the blocks produced (see §2.15 on the key-management scheme in the ADS).

Before registering with the Pool Registry, the pool operator must first register as a staker in the Staker Registry. When signing-up in the Staker Registry, the pool operator must bond some minimum coin amount (see §2.10 for details). Furthermore, the pool operator must maintain an adequate ratio of stake contributions to self-bonded stake (see §2.10 on self bond requirements for details).

- ***Tear-down***: No graceful pool tear-down functionality has been built into the Pool Registry, to simplify the state-space of the contract. Instead, if a pool operator no longer wishes to a operate a pool, they should:
  - Send a message using the meta-data protocol that they will no longer be an active pool.
  - Withdraw their self-bonded stake (violating the pool invariants and putting the pool into a *broken* state.
- **Management functions**:
  - ***Change fee***: The pool operator is allowed to change the fee charged for operating the pool, due to market conditions, etc. Note that the fee change takes effect immediately (as soon as the fee-change message is received by the contract by the pool operator). Upon fee-change, delegators take appropriate measures (e.g. stay delegated to the pool, transfer their delegation to another pool, etc.).
  - ***Update meta-data***: The pool operator is allowed to change the metadata of shown in the wallet by updating the content hash and/or the metadata URL. Since the meta-data only contains display information about pool, this feature can be used by pools to communicate updates and announcements to the delegators.
  - ***Update bonded-stake***: The pool operator can send or withdraw bonded coins towards satisfaction of self-bond requirements (see §2.10 & §2.11 for details).
  - ***Update block-signing address***: If the pool operator suspects their block-signing key is compromised, they can update that "hot" key (see §2.15 for details).

## 2.7 Operational Requirements

Since the pool operator is the entity that gets delegated staking rights from users of the system, the pool operator's primary obligation with respect to the ADS protocol is to run a Aion-Unity full node, that is well connected to the blockchain network, in order to participate in the staking protocol as a block producer. In order to satisfactorily fulfill this obligation, the operator must run computer hardware with comparable or better specifications than:

- Intel i7 (Skylake, 6th generation) processor with 4 cores, 8 threads.
- 16 GB of DDR4 RAM
- 512 GB SATA SSD
- 50Mbps dedicated internet connection

The operator is required to keep at-least 99.9% ("three nines") availability, which corresponds to at-most 8.77 hours of down-time per year. Pool operators are encouraged to host a web page, advertising self-reported up-times and hardware specification, among other pertinent information about pool operations, to instill confidence in and advertise the operator's operational capabilities.

## 2.8 Metadata Protocol

When an operator registers a new staking pool, the new pool gets added to an on-chain registry of all active staking pools. During registration, the pool operator is required to provide metadata about the pool (e.g. logo, web-page, human-readable name, etc.). This metadata could then be consumed by user interface enabling stake delegation in Aion-Unity.

We define an explicit protocol for metadata management, which involves both on-chain and off-chain actions. This protocol standardizes the way any user interfaces enabling delegation on Aion-Unity can retrieve rich data about staking pools, which empowers both pool operators and delegators:

- The pool operators have a transparent and easy process to maintain rich contextual descriptors about their staking pools, which automatically get pulled int, and updated across all user interfaces implementing the ADS. Without such standardization, staking pools would have to manage relationships with all relevant ADS front-end providers in order to get listed and service contextual-information requirements (e.g. logo, pool descriptions, etc.)

- The delegators can rely on a rich set of descriptors provided by the pool operators, widely available across all user interfaces implementing the ADS, providing meaningful data points to inform their delegation decisions.

When a staking pool registers to the ADS, the pool operator must provide a *metadata URL* and the *metadata content-hash*. The pool operator must host a JSON file at the metadata URL (HTTPS over TLS), with the following schema:

- ***Schema version***: A version number, to identify the schema. This is here to enable upgradability.
- ***Logo***: A thumbnail containing the logo of the pool. The image must be base64 encoded PNG, with the dimensions of 256 pixels-square.
- ***Description***: A "tell me about yourself"-style, short description for users to consume when making stake delegation decisions. This field shall not exceed 256 characters.
- ***Name***: A human-readable name for the pool. This field shall not exceed 64 characters.
- ***Tags***: These serve as keywords for any search functionality to be exposed by any ADS user interface. This is a JSON array. The size of this array shall not exceed 10 elements, with each element not exceeding 35 (valid) characters.
- ***Pool URL***: This is a URL, pointing interested delegators to the homepage of the pool, for additional information to peruse, in order to help make their delegation management decisions.

The JSON must be valid according to the RFC 7159 JSON specification [Bra14]. The hash of the JSON object must match the content hash provided. The document must be less than $1024 \times 1024$ bytes (1 mb). All characters must be UTF8 encoded. The document hosting service must guarantee three nines availability (99.9% uptime). The Blake2b hash of the JSON object must match the content hash provided on-chain.

We further propose the use of *proxy servers* to cache the list of staking pools and their associated metadata. This enables rapid implementation of ADS user interfaces, since one could query a web-service to retrieve this list, as opposed to querying each metadata URL in the staking Pool Registry. The implementation of this server is very simple:

- *Pool metadata caching*: When an operator updates the metadata hosted at the metadata URL, they must also update the metadata content hash on-chain. The proxy server polls the

Pool Registry, listening for changes in content hash. When the content hash changes, the cache is invalidated and the new metadata is loaded into the cache.

- *Security screening*: The proxy server shall implement rules to filter out malicious content in the metadata. As attackers evolve mechanisms to attack the user interfaces, this simple server can be adaptively updated quickly in response.

To address concerns around centralization and censorship of pool lists by the proxy server, we will open-source the proxy server implementation and encourage community participants to run these proxy servers. Furthermore, we require any ADS user interface to be configurable to query a number of these proxy servers.

**2.8.0.1 Alternative approaches** An alternative approach considered in the design of the metadata protocol was to implement the Pool Registry off-chain. There were centralization risks associated with this approach, namely the censorship by registry-maintainers of pools on this list (either by omission or manipulation of rankings).

Another approach considered was storing the metadata directly on-chain (and doing away with the metadata-URL and proxy server scheme). This strategy was not selected due to flexibility concerns. The metadata is a field primarily used by pool operators as an advertisement avenue; this schema can conceivably be required to adopt additional fields to improve richness of pool metadata. In addition, this scheme allows for the metadata to become much larger than currently specified, without concerns of block size restrictions or on-chain data bloat.

Although this operator-hosted metadata scheme increases the implementation complexity slightly on part of the pool operators, it opens-up the opportunity to improve the quality of service of the ADS user interfaces in the future.

## 2.9 Rewards-splitting Scheme

Rewards sharing must be an on-chain, automatic process that does not require any action on part of the pool operator. For each block that a pool "wins", the block rewards are split between the operator and the delegators using the following scheme: first the operator fee is deducted from the block rewards, then the remaining balance is split between the delegators, weighted by their respective stakes pledged to the pool. See §2.14.1 for details regarding how this rewards splitting is implemented.

For example, if the block reward is 5 coins, the pool fees are 20% and there are three delegators, whose pledges account for 50%, 37.5% and 12.5% of the pools total stake respectively. Then the rewards distribution (for that block) will look like the following:

- Operator rewarded 1 coin (20% of 5 coins)
- Delegator #1 gets 2 coins (50% of the remaining 4 coins, after deducting operator fees)
- Delegator #2 gets 1.5 coins (37.5% of the remaining 4 coins, after deducting operator fees)
- Delegator #3 gets 0.5 coins (12.5% of the remaining 4 coins, after deducting operator fees)

There are several mechanisms available to the operator and delegator to claims rewards from the Pool Registry contract:

- After some amount of rewards have accumulated, one can send a withdraw transaction to the ADS contract from the appropriate account (rewards account for the pool operator, coin-owning account for the delegator) to retrieve their coins as "liquid balance".

– One can also manually re-delegate the rewards-pending coins to increase the total stake delegated.
– Lastly, one can enable the auto-rewards delegation feature, which allows the rewards to be automatically re-delegated upon payout (rewards do not accumulate in the rewards-pending state in this scenario).

## 2.10 Self-bond Requirements

We propose imposing a self-bond requirement on all stakers registered in the Staker Registry. If the self-bond goes below the prescribed minimum, then all stake bonded to the staker becomes ineffective (i.e. loss of influence in PoS). There are two major reasons to implement such a scheme.

First, this creates a minimum cost (outside of the cost to run computer hardware) for someone to become a staker. This adds a barrier for anyone looking to launch a sybil attack (a single stakeholder creating a large number of pools in the hopes of dominating PoS voting power by means of random delegators choosing their pool).

Second, this facilitates the (potential) implementation of a slashing mechanism[1] to punish any misbehaviour (protocol deviation) on part of the pool operator. If there were no self bond requirement, the pool operator has no "skin in the game", and could behave maliciously (w.r.t the protocol) with impunity.

**2.10.0.1 Pool Operators** Pool operators should earn staking rewards for their self-bonded stake, just like any other delegator in the pool. Therefore, the pool operator's self bonded stake must be delegated through the Pool Registry contract.

## 2.11 Restricting Pool Size

Designing incentive mechanisms that promote decentralization in PoS delegation protocols is an open problem. PoW has a tendency to centralize via the creation of mining pools; in the Bitcoin network, mining pools have been observed to control over 50% of network hash-rate on occasion [RJZH19]. The design of a public stake-delegation protocol (ADS) for Aion-Unity requires mechanisms to incentivize decentralized behaviour, to avoid some of the pitfalls PoW mining pools have encountered.

The challenge that lies at the heart of such a design is the inherent trade-off between efficiency and decentralization:

– *Efficiency* is optimized if only one pool operator exists that all stakeholders delegate to. This would minimize the operational costs of the network (since all the stake is delegated to one node, which would be the only PoS node operating on the network); this would in-turn maximize profits for all stakeholders.
– *Decentralization* is maximized if every single stakeholder runs their own node to contribute to PoS network security. Note that the PoS subsystem in Unity was designed to boost this kind of maximally-decentralized network configuration.

Operation of the PoS network in either of the aforementioned regimes is neither desirable, nor realistic (today). Therefore, an incentive-based mechanism needs to designed to find a balanced solution, such that some large number of pools with uniformly distributed stake-delegations can be encouraged.

[1] slashing mechanisms will be considered in subsequent protocol updates

Brunjes et al. propose a rewards-distribution function such that some target number of stake pools can be achieved (proof for a Nash equilibrium arising from rational play for such a condition is provided) [BKKS18]. Although promising, further analysis is required to adapt this rewards mechanism to Unity consensus, and may be considered in an upcoming update.

Instead, the following simple scheme (inspired by Tezos [Goo14]) is proposed. It involves imposing a self-bond requirement on staker, which proportionally determines the limit on the stake that can be delegated to the pool. Consider an example: assume that the self-bond requirement is 2% and the minimum self-bond requirement is 100 coins. If the pool-operator has self bonded 1000 coins, then the maximum stake that can be delegated to this pool is $50,000$ coins.

This creates a restriction the the size of the pools, to discourage the formation of very large pools (since the pool operator would have to put a large amount of self bonded stake in order to operate a pool with large contribution margins. Note that the pool cannot be over-delegated. Any operation that delegates more stake to a pool than allowed by the self-bond-to-delegation-capacity ratio, will fail.

Although this scheme does not provably deter centralization, it produces some barriers to sybil attacks and pools becoming too large, while enabling a large number of honest pools to operate in the PoS ecosystem.

## 2.12 System Parameter Summary

The following is a summary of the system in the Aion-Unity implementation (to be updated in a subsequent draft):

- *Self-bond minimum*: This refers to the minimum amount of bond required to be "put up" by a staker in the Staker Registry. See §2.10 for details.
- *Self-bond percentage*: This refers to the minimum amount a stake a pool operator is required to have bonded, as a proportion of the total stake delegated to the pool (see §2.11 for details).
- *Post-active (thawing) period*: This refers to the time (in blocks) that stake is immobile and ineffective (w.r.t. PoS staking) after a staker has initiated a stake withdrawal (either via an undelegation or unstake operation). See §2.4 for details.
- *Transfer lockout period*: This refers to the time (in blocks) that stake is immobile and ineffective (w.r.t. PoS staking) after a staker has initiated a stake transfer to another staker. See §2.4 for details.
- *Staking (per-block) rewards*: This refers to the computation of the total per-block rewards disbursed to the proposer of a PoS block. This mechanism will be discussed in an upcoming section of this paper.

## 2.13 User Interface Concerns

In the following section, we don't define any implementation guidelines; but rather we specify some key features that the user interface must employ in the implementation of the public delegation system (ADS).

### 2.13.1 Pool Registry Presentation

The user interface shall produce a list of all active and retired pools in the stake Pool Registry. For each pool any pertinent information required by the user in order to make delegation decisions should be presented (e.g. fee, capacity remaining, rewards estimates, etc.). The user should be able to delegate to multiple pools and view all their outstanding delegations and rewards earned from each delegation (at the block-level resolution).

The user interface can retrieve the Pool Registry and associated metadata from the metadata-proxy server (as defined in §2.8). If the metadata is malformed (i.e. any of the metadata rules defined in §2.8 are violated), the metadata will be unavailable on the proxy servers; UI should be designed to handle such scenarios. Furthermore, the UI should be configurable to query a number of metadata-proxy servers to promote diversity in metadata-proxy server providers.

In order to help users make a rational decisions with respect to their stake delegations, we propose that the Pool Registry listing should be default-sorted using some weighted function of:

- the fees charged by the pool,
- the apparent performance of the pool (see §2.13.2), and
- the remaining contribution capacity.

The goal of this proposed "attractiveness score" is to promote pools that are reliable, have not yet reached saturation, and have a low cost. Further research is required to specify this function precisely.

### 2.13.2 Calculating Apparent Performance of Pools

The wallets should report some notion of up-time for a pool; this measure is critical to gauging the reliability of a pool, and directly impacts the rewards a delegator can expect to receive by delegating to this pool (delegators should rationally choose pools with the highest possible historical up-times, since even if a pool offers low fees, a spotty up-time track-record will manifest itself in diminished rewards).

Since there is no explicit way to capture an up-time metric in the design of Aion-Unity PoS (due to it's stochastic nature), we instead propose a simple solution to find a proxy for the "onlineness" of the pool operator that we are calling *apparent performance*.

A pool is considered *established* if it has been active for at-least 1 week ( $60,480$ blocks). For any established pool with sufficient size (in terms of delegated stake), we can effectively infer some notion of up-time. The reason that an inference is the best we can do is because the stake amount can fluctuate over time and the rewards are unpredictable (distributed stochastically) at every block, with no protocol-defined mechanism to measure "onlineness" of a pool operator. The apparent performance measure can be constructed as follows:

- First, we define a moving window of $60,480$ blocks ( 1 week) over which we define the following averaged metrics.
- Since the stake amount can fluctuate over time, to get a stable measure for the amount of stake delegated to a pool over a period of time, we take some average (either over the complete

interval or with gaps) of the active stake delegated to a pool over the last 1 week's worth of blocks.
- Then perform a similar calculation as above, except over the stake delegated to all the pools, to get an average for the total stake delegated to the system over the last week.
- With these two averages in hand, we can determine the expected ratio of blocks this pool should have produced in proportion to the total blocks produced over the last week.
- Finally, we need to find the ratio of the expected blocks produced over the last week to the actual blocks produced, which will give us some measure of apparent performance (a number between 0 and 1).

There are several factors that could skew this calculation. First of all, the notional 1 week might not be a long enough time over which to compute these averages. Furthermore, large swings in stake contributions could skew the computation of the arithmetic mean; this may potentially be fixed by sizing the window as a function of the standard deviation of the time-series function of stake contribution magnitudes.

When a new pool is created, there is no data to determine it's apparent performance. New pools should be distinguished from the established pool (e.g. displayed in a separate section of the UI), since no reasonable measure for future performance can be inferred.

### 2.13.3 Pool Life-cycle Notifications

The UI is responsible to produce notifications for all key life-cycle events for the pools a user has delegated staking rights to, to enable a user to make appropriate delegation decisions.

**2.13.3.1 Management Actions** The UI must notify a user when a pool goes into the *broken* stake and changes it's fees (§2.6). Any *broken* pools must be clearly identified. The user should be able to transfer any delegations from a *broken* pool to an active pool at any time.

**2.13.3.2 Inactive and Underperforming Pools** The user should monitor the attractiveness score (§2.13.1) of all pools one has delegated stake to, in order to notify the user of any significant changes in this metric; particularly, large drops in this metric implies one or more of the following things:
- A large amount of stake has left the pool.
- The pool's average up-time has dropped significantly (operator has stopped producing blocks).
- The pool operator has hiked up the fee significantly.
- The rewards earned by the user have diminished significantly.

This way, if the pool ceases to operate without being properly retired, it's delegators will be incentivized to transfer their delegation to another pool.

## 2.14 Smart Contract Design

Here, we outline the design of the smart contract implementation of the staking and delegation system in Aion-Unity. The system consists of three distinct contracts (with their relationship depicted in Figure 3):

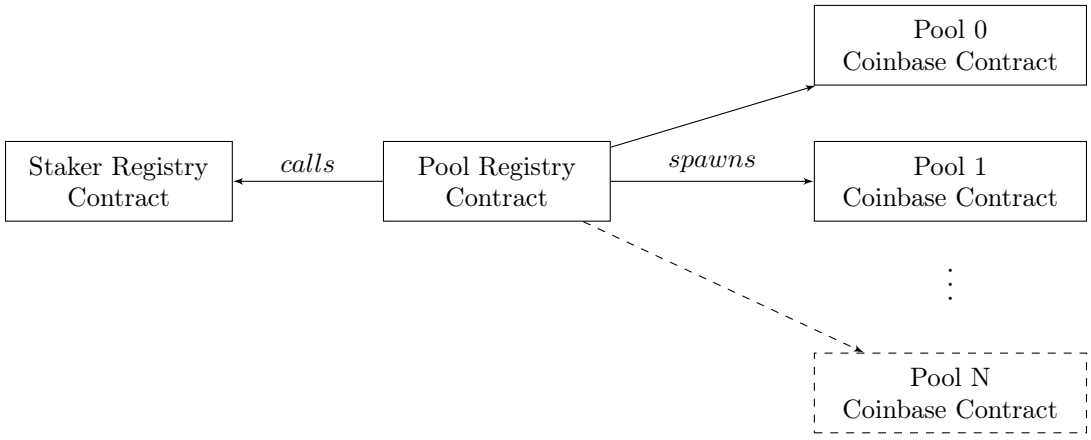


Figure 3: Smart contract architecture for stake delegation in Aion-Unity

- ***Staker Registry Contract***: This contract tracks all stake and stakers in the system and is core to the Aion-Unity protocol (i.e. this is a privileged contract, whose state is required by the protocol to resolve PoS consensus). Anyone wishing to cast their coins to stake (including other contracts, e.g. delegation contracts), need to register here. Key indirection (as described in §2.15) is built-in here; when one signs up to be a staker, one needs to provide their coinbase and signing addresses.

- ***Pool Registry Contract***: This contract is non-core to the system (i.e. has no special privileges in the system). This is the implementation of the Aion Delegation standard (ADS) and is responsible for:
  - keeping a registry of all staking pools and associated meta-data,
  - receiving coins (as delegation to a pool) and casting them to stake,
  - managing internal state of each pool (including delegation and rewards),
  - and splitting the rewards among the pool operator and delegators.

  This contract implements the StakerRegistryListener interface, which enables this contract to be notified about changes made by a pool operator in the StakerRegistry to its signing and coinbase addresses (see §2.15.1 for details).

- ***Pool Coinbase Contract***: An instance of this contract is spawned by the Pool Registry contract for each pool it instantiates. The pool's coinbase contract receives block rewards on behalf of the pool. For a particular pool, when the operator or any of the delegators request a withdrawal (in the PoolRegistry), the coins are withdrawn from this contract by the PoolRegistry. This is an implementation artifact; the end user should never need to understand the function of this contract.

We now outline the rationale for this design: recall that the PoolRegistry has no special privileges. The standard way block rewards are disbursed is by crediting the coinbase account cited in the block header. Since all pools and their states are managed in the PoolRegistry contract, if the block rewards for all pools were paid to the PoolRegistry contract, the contract could not figure out which pool the disbursement was intended for (since no contract code gets executed by the protocol upon block reward outlay). Therefore, the PoolRegistry must spawn and manage a unique contract (per pool) to receive block rewards in the standard way, and then retrieve these rewards from this contract when withdrawals are requested by any delegator or operator.

### 2.14.1 Rewards Distribution Algorithm

One critical requirement regarding rewards sharing is that "rewards sharing must be an on-chain, automatic process that does not require any action on part of the stake pools". The problem of on-chain rewards distribution appears trivial at first glance, but deserves significant attention care in its design.

First, lets consider the naive solution to the problem of block rewards distribution. Assume some hypothetical "onBlockProduced" function that executes as the last transaction in a PoS block produced by a staking pool. In this function, the block rewards are be disbursed to the pool's operator and delegators. The time-complexity of this function is linear in the number of delegators in the pool. Since the disbursement to each delegator requires at-least one state update (database write), state updates linear in the number of delegators in the pool would need to be performed. Furthermore, since state updates are a fairly costly operation in the Aion-AVM regime, especially for large pools, updating the state for every delegator in a pool alone, could easily surpass the total gas limit of a block (which represents the maximum amount of computation that network participants can perform in the validation of blocks, while keeping the block-time stable). Since the combined gas cost for disbursements and the included transactions in a block cannot exceed the total block gas limit, this naive solution is clearly impractical: consider a "full" PoS block; there is no gas budget left (in the block) for rewards disbursement!

**2.14.1.1 F1 Rewards Distribution** A clever solution this problem was proposed by Ojha [Ojh19]. A detailed description of the rewards distribution mechanism is out of the scope of this report (please refer to [Ojh19] or a toy-implementation by the author [Sha19]).

This algorithm moves all rewards distribution computation from some hypothetical "onBlock-Produced" function (which is not available in the Aion protocol to begin with), to any events where the rewards-per-unit-stake changes (e.g. delegate, undelegate, withdraw, etc.). The rewards distribution is still linear in the number of delegators, but the computation is amortized over the interactions users have with the system (i.e. all fee distribution computations are "fuelled by" user-provided gas).

Furthermore, this algorithm is approximation-free; in the rewards distribution calculations, the only source of approximation is finite decimal precision, which cannot be avoided (confirmed using simulation in [Sha19]).

### 2.14.2 Asynchronous Tasks

There are several instances of "asynchronous" tasks in the Aion-Unity staking and stake delegation systems:

- ***Unbonding Stake***: Upon unbonding or undelegating of stake, the coins must be ineffective in PoS and immobile (not liquid) for at-least the thawing period §2.4.
- ***Transfer Stake***: Upon transfer of stake from one staker to another in the Staker Registry contract, the coins must be ineffective in PoS and immobile (not liquid) for at-least the transfer-pending period §2.4.
- ***Auto Rewards Delegation***: Rewards earned by delegators in the system can be "auto-delegated"; i.e. coins earned as part of block rewards can be automatically cast to stake on behalf of the delegator.

These features are asynchronous in the sense that these interactions cannot be completed within one transaction initiated by the user; they require some action by the protocol itself (not initiated by a user), after some condition on contract or chain state has been met.

In the case of unbonding and transfer stake, they are examples of time-locks, which require an action from the system, delayed into the future from some trigger-action. On the other hand, auto rewards delegation (in it's most trivial incarnation) is a case of the protocol taking an action on behalf of the user upon an event (disbursement of rewards).

No mechanism exists in the protocol for transactions to be scheduled in the future or be automatically triggered if a (chain or contract) state change occurs, since it violates security constraints of the AVM computation metering system.

Therefore, the aforementioned features must be implemented vis à vis two disparate transactions: an initiating transaction and a finalizing transaction. The initiating transaction is sent by a user looking to affect their state (unbond, transfer or enable auto rewards delegation). The finalizing transaction can be performed by anyone in the system (including the user himself) in order to complete the initiated action.

We considered incentives that people would have to make finalization calls on other users' behalf. The following mechanisms were considered:

1. The finalization function could be called by the initiator of the asynchronous transaction.
2. If the asynchronous transaction involves movement of coins (e.g. auto rewards delegation), one could attach a fee that callers of the finalization function could collect. The idea is that bounty-seekers could scrape accounts registered for this finalization function; they would wait for enough coins to be accumulated such that the fee collected upon a function call exceed the caller's transaction cost by some profit threshold.
3. The staking pool could be expected to perform the transaction on behalf of it's delegates (this service could be included in the fees charged by the pool). In this case, we propose the following extension to the scheme: the ability to batch multiple finalization calls into a single transaction, to minimize the total transaction numbers on the network.
4. The transaction initiator could provide a nominal amount of coins as a bounty when sending the initial transaction in the asynchronous transfer, which could be collected by the caller of the finalization transaction

In the case of auto rewards delegation, strategy 2 is recommended. In the case of the time-lock interactions (transfer stake, transfer delegation, unbond, undelegate), strategy 3 is recommended, with the possibility of a sufficiently motivated user calling the finalization transaction (as a last resort), since no restrictions are placed on the caller of finalization transactions.

## 2.15 Key Management

The staker and pool registries disperses system-responsibilities over several asymmetric key-pairs, to carefully refine the security requirements on each key, which is standard practice in industry.

**2.15.0.1 Staker registry symmetric-key rights** A (private) staker required to manage the following five (5) keys, each of which correspond to a distinct function:

- *Identity key*: The address corresponding to this key becomes the identity of the staker in the Staker Registry system. This key cannot be used to perform any management actions in the context of the Staker Registry. Each identity key can only be used once to register a staker (since identities are unique in the Staker Registry). There is no mechanism to replace this key with another one.
- *Management key (cold key)*: The address corresponding to this key is recorded upon staker registration and there is no mechanism to replace this key with another one. This key should be kept in cold storage (HSM) at all times, and only be retrieved when management tasks need to be performed. This key has the following management rights:
  - Change registered coinbase address
  - Change registered block signing address
  - Change registered self-bond address
  - Set the active status of the poo
  - Attach and remove any (Staker Registry) listeners
- *Block-signing key (hot key)*: This key entitles it's holder the rights to produce blocks on behalf of ala stake delegated to the staker. This key needs to be kept online while connected to the Aion-Unity network, since this key is required to sign any blocks won by the staker in the course of the PoS "lottery". At the moment, the guidance is to load this private key in the memory of an appropriately permissioned process, or in a commercial HSM (e.g. YubikeyHSM 2) that supports EdDSA (ED25519) signatures. If this key is compromised, the attacker can censor (deny inclusion of) transactions within the blocks supported by the delegated stake. Furthermore, block signing key can't be changed too fast (subject to a cooling period) to avoid a variant of the stake grinding attack (as described in [WZS19]), where the variable that an attacker can "grind" is the block-signing key to give the attacker an advantageous time-to-next-block.
- *Coinbase address*: PoS block rewards get paid out to the address corresponding to this key. If the address corresponding to this key is a smart contract, then it must be secured from known attacks on contract balances. If this address is a user account, the receiver of the rewards must secure this account appropriately.
- *Self-bond key*: The address corresponding to this key must bond enough balance to satisfy Staker Registry's self-bond requirements §2.10.

#### 2.15.0.2 Pool operator symmetric-key rights

The pool operator is required to manage two private keys, each of which corresponds to a distinct function in the operation of the pool:

- *Block-signing key (hot key)*: This key is functionally equivalent to the corresponding key described in the section above (§2.15.0.1). Same recommendations and guidelines apply. This key must be provided upon instantiation of the pool.
- *Management key (cold key)* This is the key used by the operator to register the pool. This key entitles it's holder the rights to all pool management tasks §2.6. This key should be kept in cold storage (HSM) at all times, and only be retrieved when management tasks need to be performed. There is no mechanism to replace this key with another one, since this (public) key is the identity of this pool; this public key is used by any entities addressing this pool for any on chain interactions with this pool (e.g. delegators use the address corresponding to this key-pair to identify the pool they would wish to delegate their staking rights to).

  If this key is compromised, the pool is compromised and must be shut down. If this key is compromised, the attacker can take over pool operations (e.g. shut-down the pool, etc.). It is important to note that even if this "master pool key" is compromised, the delegated stake is in no risk. To reiterate, no key, except for the user's very own, has rights to alter any votes or withdraw any bonded coins. In the event of a management key compromise, the user must simply transfer delegation rights to another pool with sufficient delegation capacity to resume any interruptions in rewards outlays.

Note the pool operator is shielded (by the Pool Registry contract) from much of the complexity of managing the extra keys associated with the operation of a private staker through the Staker Registry.

Note that when registering for a pool (§2.6), do not use exchange addresses for any of the keys. The operator needs to control the private keys for these accounts in order to perform pool actions such as retrieve the rewarded coins, sign the blocks, perform management actions, etc.

### 2.15.1 Pool States: Broken and Active

The staking pool life-cycle is very simple, as depicted in Figure 4. A staking pool can either be in the *active* state or the ***broken*** state. If the staking pool in the active state, it implies it is eligible to receive delegated stake and fulfill all responsibilities of a pool-operator within the protocol. The pool starts in the *broken* state and remains so until the pool has been successfully initialized (via fulfillment of self-bond requirements). Furthermore, if any of the pool invariants are violated (see §2.14), the pool is set to the broken state. Note that any coins still delegated to a broken pool will still be effective as stake in the system until undelegated or transferred.

The pool is in the *broken* state if any of the following conditions are true:

- If the coinbase address in the Staker Registry is set to anything other than the address of the coinbase contract deployed by the Pool Registry contract on pool initialization.
- If the Pool Registry contract is removed from the listener list for this pool.
- If the pool's self-bond requirement in the Staker Registry is violated (see §2.10).
- If the pool's self-bond percentage in the Pool Registry is violated (see §2.10).

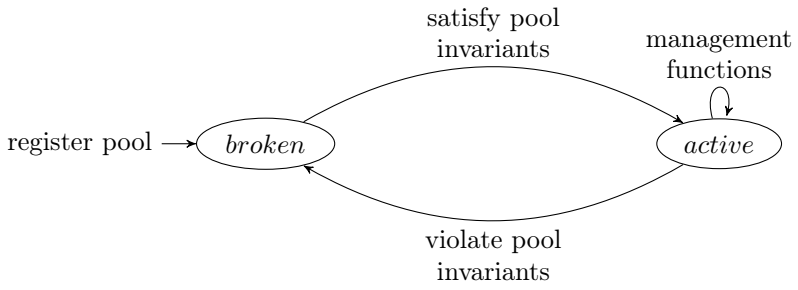


Figure 4: Staking pool life-cycle

In the current design, the only way a pool can violate an invariant is by violating the self-bond requirements (since the other requirements are maintained by the Pool Registry contract). The repercussion for leaving a pool in the *broken* state (imposed on the pool operator) is the inability to generate blocks using pooled stake.

## 2.16 Protocol (Kernel) Responsibilities

Since the function of the protocol depends on the StakerRegistry contract's state, there must be some special interactions that must to coded into protocol implementation (i.e. "the kernel").

Every time a PoS staker (node operator) produces a block, the kernel must look-up the staker by his signing key (§2.14) in the StakerRegistry to validate the following:

- the claimed time-to-next-block is valid, based on their committed stake in the StakerRegistry,
- the claimed coinbase (from the block header) is the registered coinbase in the StakerRegistry,
- and the staker is eligible to produce a block at the proposed block height.

The enforcement of self-bond requirements is implemented in the Staker Registry contract, so all the kernel needs to do is to call the following method,

$$getEffectiveStake(signingAddress, coinbaseAddress),$$

which returns the effective stake after taking stakers self-bonded coins into consideration.

# References


[BKKS18] Lars Brünjes, Aggelos Kiayias, Elias Koutsoupias, and Aikaterini-Panagiota Stouka. Reward sharing schemes for stake pools. *arXiv preprint arXiv:1807.11218*, 2018.

[Bra97] Scott Bradner. Rfc 2119: Key words for use in rfcs to indicate requirement levels, 1997.

[Bra14] T Bray. Rfc 7159: The javascript object notation (json) data interchange format. *Internet Engineering Task Force (IETF)*, 2014.

[Goo14] LM Goodman. Tezos—a self-amending crypto-ledger white paper. *URL: https://www. tezos. com/static/papers/white_ paper. pdf*, 2014.

[KBC19] Philipp Kant, Lars Brunjes, and Duncan Coutts. Engineering design specification for delegation and incentives in cardano–shelley. *IOHK Blog*, 2019.

[Ojh19] Dev Ojha. F1 fee distribution. Cosmos SDK Documentation, 2019. URL:https://github.com/ali-sharif/f1-fee-distribution/blob/master/Ojha19.pdf (version: 2019-01-03).

[RJZH19] Matteo Romiti, Aljosha Judmayer, Alexei Zamyatin, and Bernhard Haslhofer. A deep dive into bitcoin mining pools: An empirical analysis of mining shares. *arXiv preprint arXiv:1905.05999*, 2019.

[Sha19] Ali Sharif. F1 toy implementation. Github, 2019. url:https://github.com/ali-sharif/f1-fee-distribution/ (version: 2019-08-02).

[WZS19] Yulong Wu, Yunfei Zha, and Yao Sun. A unifying hybrid consensus protocol, 2019.

[YZW19] Ge Zhong Yunfei Zha and Yulong Wu. Aion-unity proof of concept report. Aion Technical Documentation, 2019.

# Appendices

## A System Smart Contracts

Following are the AVM smart contracts (extracted as Java interfaces) for the staking and delegation contracts. Only the functions utilized in the major control flows (as described in appendix B) are provided. Also note that access modifiers (e.g. public, private, etc.) at the class and method level have also been stripped for brevity.

```
interface StakerRegistry {
    void registerStaker(Address identityAddress, Address managementAddress,
                        Address signingAddress, Address coinbaseAddress,
                        Address selfBondAddress);
    void vote(Address staker);
    void unvote(Address staker, long amount);
    void unvoteTo(Address staker, long amount, Address receiver);
    int finalizeUnvote(Address owner, int limit);
    void transferStake(Address fromStaker, Address toStaker, long amount);
    int finalizeTransfer(Address staker, int limit);
    void setSigningAddress(Address newSigningAddress);
    void setCoinbaseAddress(Address newCoinbaseAddress);
}

interface PoolRegistry {
    Address registerPool(Address signingAddress, int commissionRate,
                         byte[] metaDataUrl, byte[] metaDataContentHash);
    void delegate(Address pool);
    void undelegate(Address pool, long amount);
    void redelegateRewards(Address pool);
    void transferStake(Address fromPool, Address toPool, long amount);
    long withdraw(Address pool);
    int finalizeUnvote(Address owner, int limit);
    void finalizeTransfer(long transferId);
    void enableAutoRewardsDelegation(Address pool, int feePercentage);
    void disableAutoRewardsDelegation(Address pool);
    void autoDelegateRewards(Address pool, Address delegator);
}

interface PoolCoinbase {
    void transfer(Address recipient, long amount);
}
```

# B System Interactions

This appendix illustrates major user and cross-contract interactions, for didactic purposes. Note, that anything in curly brackets ({}) after function calls indicates the coin amount enclosed in the transaction (function call).

## B.1 Private Staking

Figure 5 describes the scenario where a Staker chooses to privately bond their coins as stake (i.e. run their own node), and then after some time, unbond those coins. The staker must first register with the StakerRegistry contract. Then the staker must send a "vote" message (to their own address in the StakerRegistry), sending the amount to bond as the transaction amount.

When the staker is ready to unbond the stake, the staker must do so across two transactions. First, the staker must send the unvote (with the amount to unvote) transaction, which commits the stake into a *thawing* state. This function returns an *unvoteId*, which uniquely identifies this unvote operation. After the thawing period has elapsed (§2.4), at any time, any user in the system can call the finalize function, to release the thawed coins back to the staker.

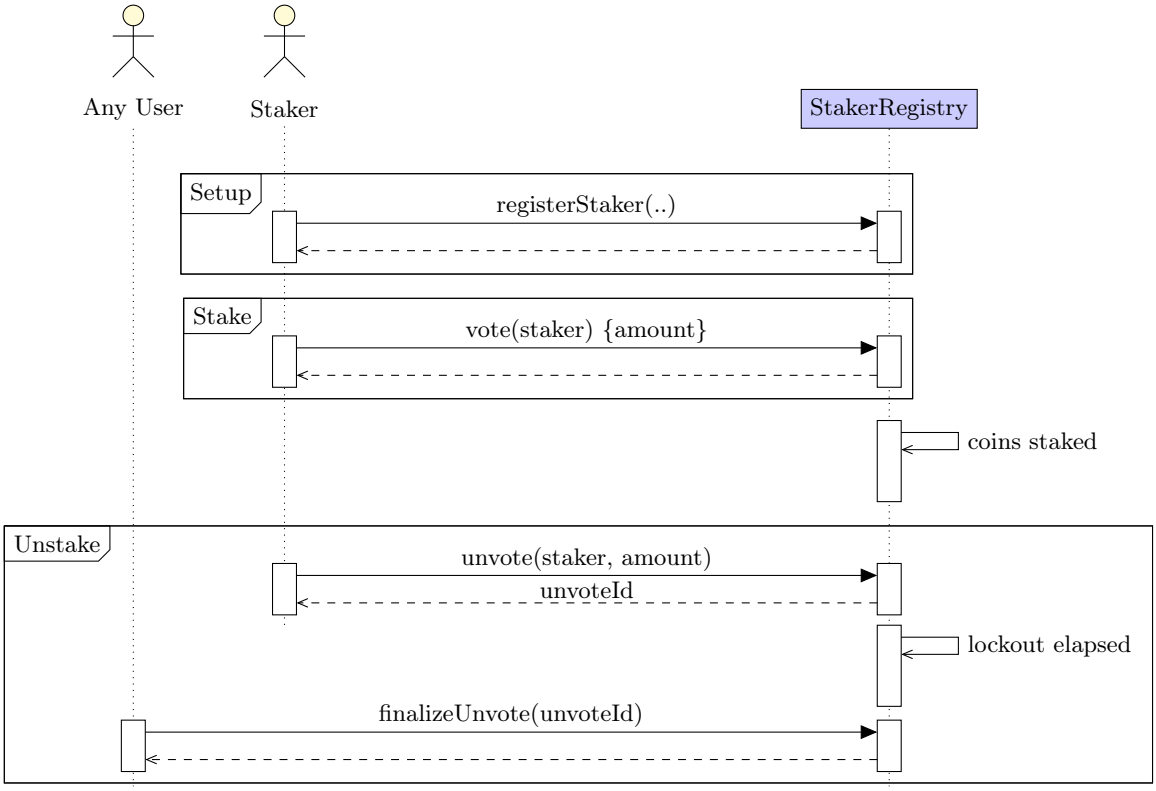


Figure 5: Private staking in Aion-Unity

## B.2 ADS Pool Setup

Figure 6 describes the setup of a staking pool. The operator must sign up with the Pool Registry (using their hot and cold keys). The Pool Registry sets up the Pool Coinbase and Pool Custodian contracts and registers the pool as a staker in the Staker Registry contract.

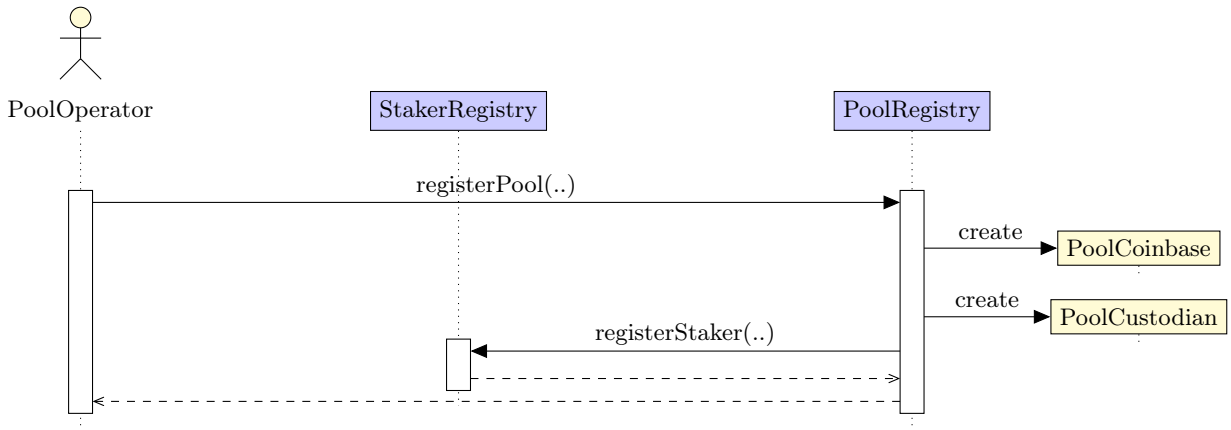


Figure 6: ADS pool setup in Aion-Unity

## B.3 Delegation to an ADS Pool

Figure 7 illustrates the mechanism of delegation and undelegation of stake to an ADS pool. Note that the delegator only has to interact with the PoolRegistry contract to complete their delegation workflow.

In order to delegate stake, a user must send their coins to the PoolRegistry in a delegate transaction. When delegate is invoked, the PoolRegistry records the delegation and adds the stake to the pool's balance in the StakerRegistry; to the staking registry, the pool looks like one big staker.

The undelegation of stake is a two-step process (since unbonding of stake is involved). When a user calls the undelegate function in the PoolRegistry, an unvoteTo is triggered in the StakerRegistry, which returns the corresponding unvoteId, which uniquely identifies the thawing of this parcel of stake. After the thawing period has elapsed, any user can call the finalizeUnvote function, either through the PoolRegistry or directly in the StakerRegistry with the unvoteId of the delegator, to release the liquid coins back to their account.

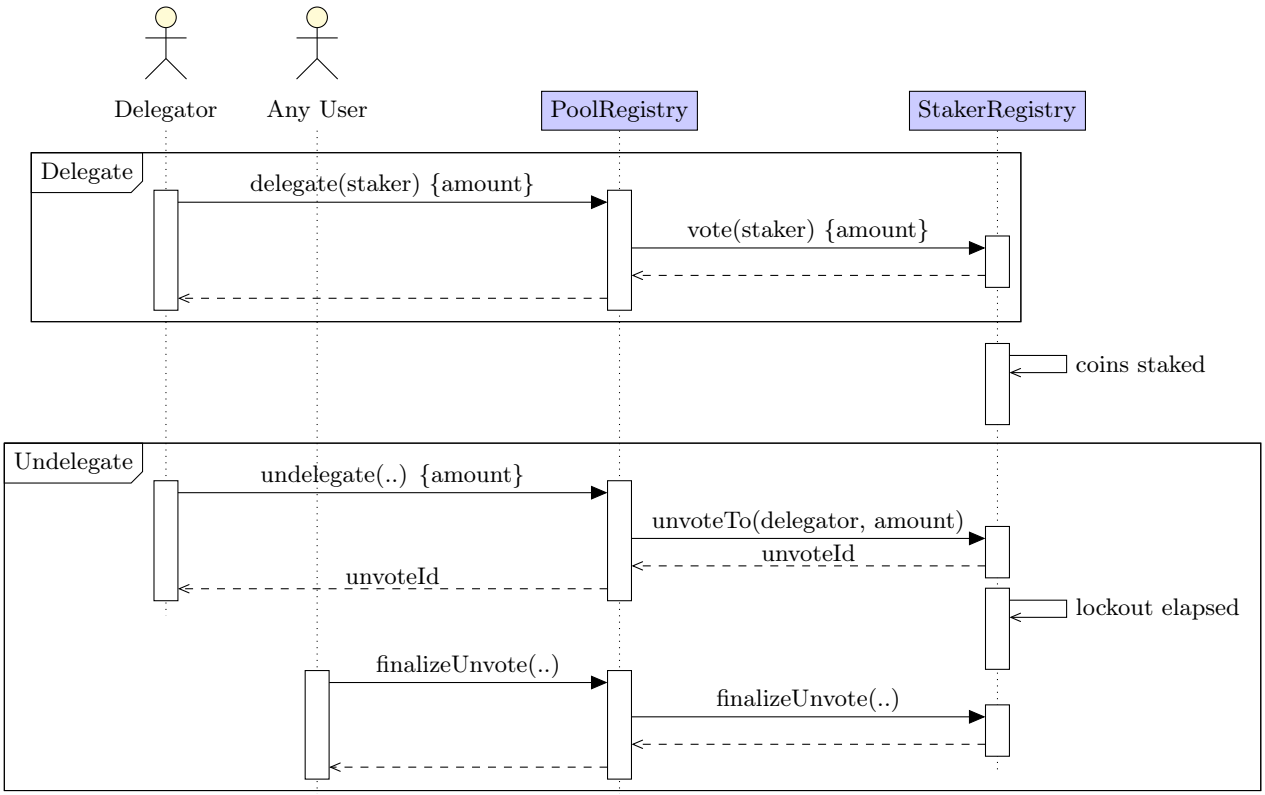


Figure 7: ADS delegation and undelegation in Aion-Unity

## B.4 Auto-Delegation of Rewards

Figure 8 demonstrates the auto rewards delegation feature in ADS. A user can enable this feature either at the moment of a delegation, or after the fact as a separate transaction. Once enabled, this feature enables any user to call the auto-redelegate function to commit a delegator's earned rewards as stake. At any point, the delegator can disable this feature.

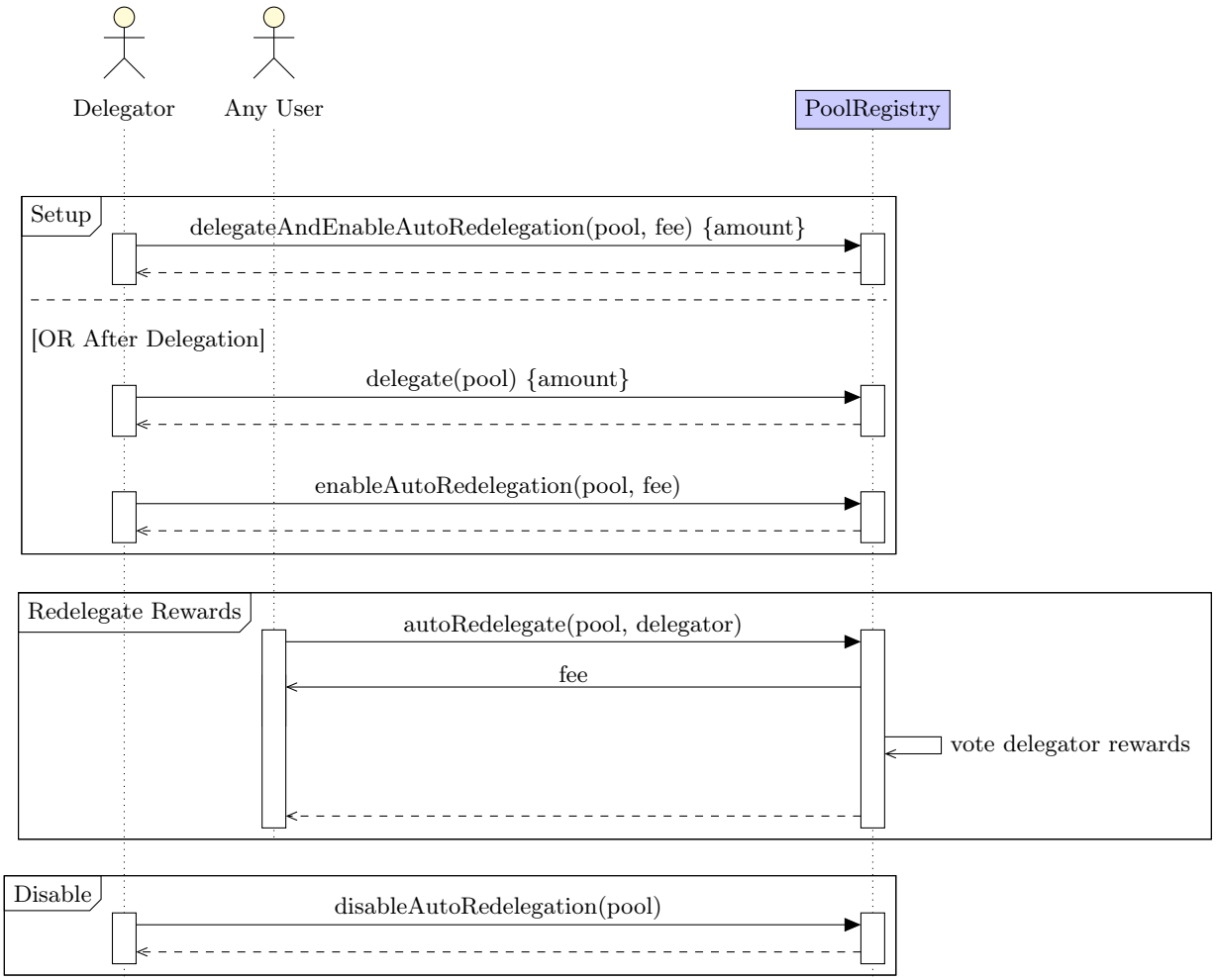


Figure 8: Auto redelegation of rewards in Aion Unity

## B.5 Stake Transfer

Figure 9 demonstrates the stake transfer feature, at the level of the PoolRegistry (although this interaction looks almost identical for a solo-staker interacting directly with the StakerRegistry).

A delegator must initiate a transfer of stake between stakers (pools) at PoolRegistry, which in-turn reflects the transfer in the StakerRegistry. A transferId is returned, which uniquely identifies this transfer. After the transfer lockout period has elapsed, any user can call finalize to move the stake between the source and destination stakers.

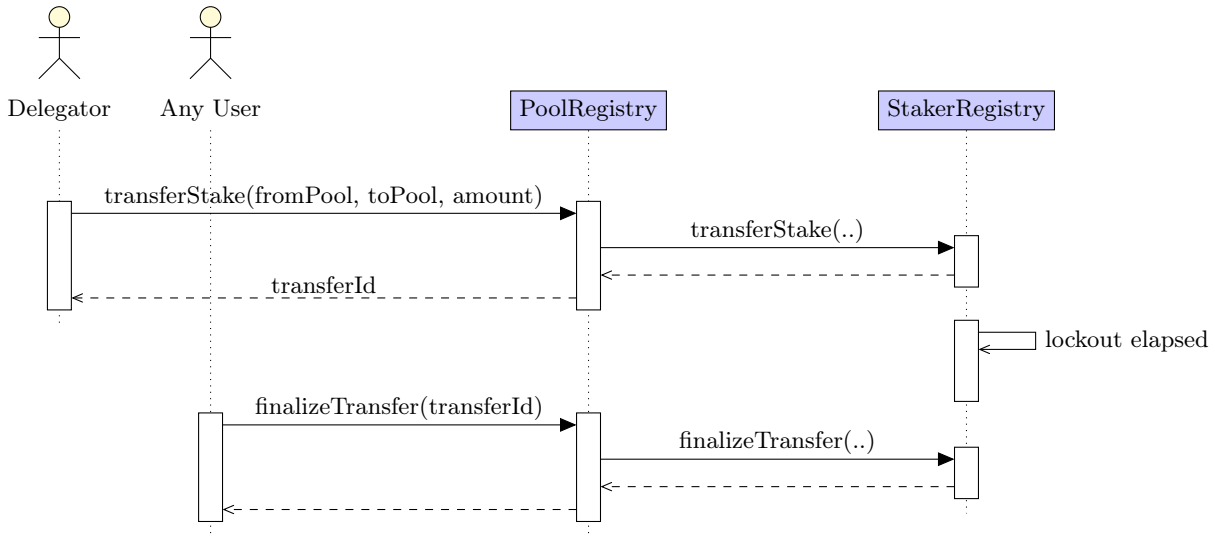


Figure 9: Stake transfer feature in Aion-Unity

## B.6 Delegator Withdrawal

Figure 10 demonstrates a withdrawal of rewards earned by a delegator. Rewards are continually withdrawn from a pool's coinbase contract any time the stake apportionment in the pool changes (via a delegation, undelegation, etc.) and managed by the PoolRegistry (see §2.14.1 for details). A withdraw is yet another trigger for the pool's coinbase to be emptied of accumulated rewards (if any exist). Then, the F1 rewards sharing algorithm is invoked to compute the rewards owed to the delegator, which are promptly disbursed before winding down the transaction.

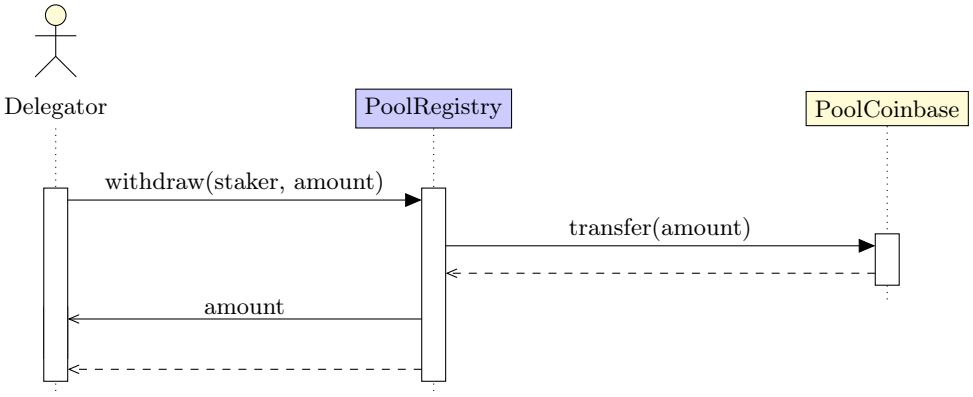


Figure 10: Delegation rewards withdrawal in Aion-Unity