Adamnite Protocol Specification
Version 0.1

April 2023

This document is meant to be a protocol level description of the Adamnite Blockchain. It is meant to supplement the white paper and technical paper to help guide potential client developers as they develop libraries that correspond to different parts of the core protocol, which includes the ledger, the gossip and networking layer, storage, and the consensus protocol. It does not include any formal discussion on the virtual machine or programming language, beyond referring to in its finalized bytecode in the context of core state transitions.

Contents

1 Distributed Ledger

Adamnite's distributed ledger is composed of an arithmetic and never-ending sequence of timestamped data structures, which we refer to as blocks. New blocks are added to the sequence through a decentralized agreement protocol, which we formalize in section 2. A section of time is referred to as rounds; this is akin to epochs or checkpoints in other distributed ledger protocols. Every round r, a set of blocks B_1 are added to the ledger. Each block B consists of a set of individual state transitions, which we call transactions (we define a state transition as an instruction or set of instruction that are executed at a set point in time), and cryptographic metadata attesting to the validity of the block.

The ledger consists of a set of accounts, a, which each maintain a balance in the native currency. An account can be owned by a user, or be a smart contract, meaning that it automatically executes transactions and message calls based on predefined logic. An account is identified by its public address, which is a unique set of bytes. A public address is generated by applying the following function to a $secp256k1$ generated public key x: $RIPMED160(SHA512-256(x))$. This refers to the truncated version of SHA-512 (first 256 bits), hashed by the RIPMED-160 hash function. Balances and account information are stored on the blockchain directly, while contract code and program storage are stored off-chain and validated on the main network as hashes. An individual account owned by a human user sends a message call, which can be a transaction or a call to an application, by signing the data bytes of the underlying message with their private key.

A block should have variable and dynamic sizes. An individual block should have a maximum size of 130 MB, with the underlying variability dependent on the demand for the network at any given point in time. The current canonical view, or state, of the overall ledger is dependent on two variables: the underlying consensus mechanism discussed in section 3, and a fork choice rule.

The fork choice rule describes how the set of validators (defined as anyone who is operating a node within the network) define their view of the current state. This allows them to further protect against potential rebalancing attacks, where a malicious block producer or witness creates a block on multiple forks and attempts to convince the rest of the network that their fork is correct. It also allows them to disrupt malicious blocks, which may include double spend transactions. Adamnite's fork choice rule is mostly derived from the Goldfish protocol, which was a proposal to mitigate various consensus-related attacks against Ethereum's nascent proof of stake network.

Most notably, Adamnite utilizes message buffering, where nodes merge all new message calls or transactions within the network (assuming that individual message calls propagate throughout the network in an efficient manner) into a localized view. Once a new state transition has been proposed, each individual node has 500ms to signal their approval of the state transition by merging with their local view and signalling to the network via a signed approval message that they agree to it. As long as 2/3 of the total potential voting power (defined as 2/3 of the total amount of coins in the network) are controlled by nodes who approve the block, it is valid.

We also use state expiry, similar to vote expiry, for defining how new witnesses update their current state. Whenever a new witness gets selected to be a part of the block proposal phase for the current round of consensus, they define their current state view as starting at the genesis block, and the most recent block/state transition (the block they will build upon) as being the block that was approved by a prior witness pool with the highest proportion of votes

ascribed to it. That is, intermittent or prior delegations of stake to a different fork or state view will have no impact on the current state.

2 Networking

Adamnite utilizes its own independent networking layer to relay consensus-related messages and enable different contracts to communicate with one another. Adamnite's networking layer is heavily focused on maintaining BAR (Byzantine, Altruistic, Rational) resiliency. The gossip protocol should be optimized for the BAR resiliency, and be a general library that allows anyone to build a protocol on top. We provide an overall general specification of how such a network may be constructed, without providing too many details into any specifics.

Each node that connects to the network has a unique node identity, which is a randomized string unique to the Adamnite network. Each node maintains a table of other nodes that it may connect with: this table comprises the other node's public *uid* (a string determining its identification within the network) and their current status. Nodes are assigned three categories based on their recent behavior: greylist, whitelist, and blacklist. Nodes with the greylist designation are neutral; they can be promoted to a whitelist or demoted to a blacklist based on their behavior during routine protocol handshakes. Nodes with the whitelist designation are the current list of peers that an individual node can connect with. Nodes with the blacklist designation are nodes that have misbehaved, by either sending incorrect information or flooding the network with nonsense information.

All consensus-related information within the Adamnite Protocol is relayed using the gossip network. This includes transactions/message calls, which can be relayed between any and all nodes at any time, and consensus-specific information, which is often restricted to a specific group of elected witnesses in a specific time. General gossip instructions should be followed: transactions and other consensus-related messages should be ordered in a way that guarantees efficient processing.

An individual node should be able to store their node communication table directly in memory; further optimization is left to the implementor of the specific client. Protocol-related messages such as ping-pong and ACK pushes should be similar to other decentralized peer to peer protocols.

Adamnite nodes store peer information in a local table. Modelling the table as a distributed hash table, we utilize the Brahms peer to peer protocol for independent peer sampling (where the set of peers received for an individual node is not dependent on a seed node). Specifially, nodes send each other a set of node ids X; each node uses random sampling to select x node ids (where x

is a subset of X) to actually initiate communication with.

Nodes are rate limited: each node can only send another node r node ids within a round; this is to prevent nodes from flooding individual peers with nonsense ids for the purpose of isolation or executing an eclipse attack. Nodes are also able to requests pulls from other nodes to get a list of their peers to connect to. The key innovation with the Brahms protocol is the use of permutation-based sampling to update an individual node's view at any given point in time. This protects against network partitions, where an individual view of various nodes is differentiated due to attacks or network churn. By using historical random samples to update individual views, and then sorting through simple ping-pong messages, a node can ensure that it is receiving updated information from active peers over time.

3 Consensus

The sequence of blocks in the distributed ledger, B, is an agreed upon set of state transitions from the genesis block, B_0. A state transition is any change to the overall state of the ledger: a transfer of the base currency, nite, from one account to another, is a state transition that is defined in the literature as a transaction, while a change in the underlying storage called by an individual account is defined as a message call. These state transitions are agreed upon through an efficient Byzantine consensus protocol, with security guaranteed as long as 2/3 of the net stake in the system is owned by honest participants.

Adamnite utilizes a variation of the Delegated Proof of Stake protocol, where individual accounts vote, or delegate, their stake to potential block producers. Consensus and block production times are split into rounds, with a set of block producers gaining the ability to produce blocks for one round each. A round typically consists of 162 blocks, with each block contains multiple transactions and message calls.

Each account within Adamnite has multiple keys/keypairs: every account has the standard spending keypair, which determines their ability to send transactions in nite to other accounts, a participation key, which is used to sign a certificate signalling the account's decision to become a candidate for the next set of block producers, and a verifiable random function (VRF) key pair used to execute VRFs. Voting transactions are delegations from an account's individual stake to another account for the purpose of signifying their approval for them to be a part of the next set of witnesses that are in charge of block proposal. A voter can delegate their current stake to any address within the ledger, and unstake their current stake at any time. New snapshots of delegation to determine the next set of block producers are taken at the conclusion of every round.

Adamnite has two sets of witnesses (we use the term witness to describe a node who takes an active role in producing the next block). The first set of witnesses, defined as chamber A, control the main pool of transaction-based state transitions, and propose blocks that are appended directly to the core distributed ledger. The second set of witnesses, defined as chamber B handle message calls, which in most cases refers to a call to a specific on-chain program or smart contract. These witnesses process both calls to existing smart contracts and the creation of new contracts. Both contract code and state is processed off-chain and stored in an efficient read-write database format. The hashes of both of these parameters are stored on-chain in the underlying state, and are changed by chamber A whenever a new batch of message calls is uploaded by chamber B.

We now describe the consensus process step by step. At the beginning of each round r, all witnesses who had any portion of nite delegated to them (defined as one unit of the lowest possible denomination of nite) execute a Verifiable Random Function that takes as input the witness' VRF private key and a random seed generated pseudorandomly for the current round, and generates an output V. We model the VRF as a blackbox pseudorandom function whose output can be verified in an efficient manner. The witness then compares their output with a separate value P, which is an overall score comprised of the amount of votes that they received, their own stake within the network, their reputation score (based on the Repustake algorithm), and the amount of times they have been selected in the past. If $V < P$, then the witness is eligible to be a part of the next round of consensus.

We formalize the process for a witness W below:

$$VRF(W_V SK, Q) = VEVALUATE(V < P)$$

(1)

In the formalization above, $W_V SK$ represents the secret key of the witness, while Q represents the random seed allocated to the round. Q is initialized at the genesis block, and salted with a random value at the conclusion of every round.

The first x witnesses (where x is a constant) who can prove that they are valid witnesses to the system (we model the system as a generalized participant; in reality, the system is simply the implementation that all peers within the protocol are using) are selected to be a part of quoroms A and B respectively.

4 State Transition Logic

Adamnite's fundamental state transition logic is an extension of the core consensus protocol; it determines how chambers A and B reach consensus on a set

of state transitions and propose them for finality.

Chamber A reaches consensus quite simply: a simple leader rotation schedule that evenly divides the number of blocks each individual is responsible for is created, and each witness gets an opportunity to create new blocks for the purpose of being included in the core blockchain. Once a block producer has created a block, the other witnesses review it, and signal their approval by signing a certificate. Once 2/3 of witnesses signal their approval, the block is approved, and is scheduled for finality as described previously. This process should take a total of 500 MS; if agreement cannot be reached in that time, then an empty block is produced. If a witness misbehaves, either by continuously proposing incorrect blocks that do not align with the state view and protocol messages of the other witnesses, by building on a fork of the current chain, or by simply being inactive for more than the allotted time (500 ms), then they are replaced with another eligible witness. If a fault is reached where there are no eligible witnesses left, then a witness is randomly selected, with the amount of votes they received being the primary weight.

Chamber B reaches consensus similarly to A, with one significant difference: the transitions themselves are processed differently. A given block producer B_p organizes a batch, S, of state transitions, providing proof that time elapsed between each by hashing their individual state view using a one-way hash function after applying each state change. The block producer then splits the batch into sub-batches, providing a proof of the state of the ledger before and after each individual state transition within the sub-batch was applied. The sub-batches are split according to their proximity; individual state transitions that have a direct impact on one another (because they call the same contract or are called by the same account) are organized into the same batch linerarly. These sub-batches are then assigned to other individual witnesses in B, who can verify them effectively due to the proofs provided. This allows chamber B to process message calls in an extremely efficient manner.

Once chamber B has finalized a batch, it is signed by the entirety of chamber B , and the corresponding certificate and compressed batch are sent to chamber A. The corresponding hashes in the state on-chain are changed by the current witnesses in chamber A, who check the batch for validity before approving it. An invalid batch can be due to malicious behavior from a chamber B witness, or a malicious transaction that is attempting to double-spend to take advantage of the two-tier process (for example, an account that spends all of their nite before a message call they made is executed). A batch is all or nothing; if an individual state transition within a batch fails, then the entire batch is declared incorrect.

5 Storage

The core ledger is a state replicator that maintains the history of the blockchain, from the genesis block to the most recent block, among the network participants. The ledger can be modeled as a set of state machines, each of which maintains a record of a specific aspect of the overall network. A block represents a transition between from state to another. The following trackers are maintained by the ledger: a core state tracker, which maintains a direct reference to accounts, balances, and hashes pertaining to code and storage, a transaction tracker, which maintains a history of all transactions and also has a reference to the most recent set of transactions that have not yet been finalized but have been committed by the witnesses, and a witness tracker that maintains a reference to all witnesses who participated and were selected any given round.

All trackers should be built for read-write efficiency: new information should be able to be added quickly, and a persistent API should exist for other services to be able to easily read from each of the tries effectively. Note that the generalized implementation can be done with any sort of database (LevelDB, SQL, etc). We recommend using LevelDB due to the existing resources available in GoLang.

The specific reference for storage should be a binary merkle trie, with the root of each trie being used to verify the correctness of the contents. A future protocol upgrade may move toward using verkle tries, which leverage vector commitments instead of hashing functions for the purpose of providing evidence of tamper-proof records.