2 Chainz: All the Blockchains

Immutable, decentralized asset ownership without consensus

Mike Audi
TIKI inc.

***Abstract* – Traditionally, asset ownership has been an expensive and slow process. Immutability and decentralization enabled by blockchain technology has created a new form of ownership. While significantly cheaper and faster, ownership of many assets, like data, remain out of reach. We propose a new blockchain based system purpose-built for high frequency, low-cost assets.**

I. Introduction

What if you built a blockchain without a single agreed-upon ledger? Without commit consensus, nodes diverge.

Node divergence is a critical issue for protocols based on an individual chain. The same request can return different results depending upon the node. Unlike systems with eventual consistency, each transaction added compounds the problem.

As opposed to the traditional pattern of one chain per protocol, we outline a new system comprised of a network of chains, each with its own ledger, purpose-built for asset ownership.

Why? Speed and cost. Distributed consensus remains a bottleneck for shared state blockchains. Removal expands the power of decentralized, public, immutable, and permissionless systems to new asset classes.

II. System Architecture

Using blockchain nomenclature, we combine the traditional chain, wallet, block assembly, and validator functionality into a self-contained node. Nodes can run in both client and server environments without the need for peer-to-peer communication.

Each node maintains its own chain, with blocks signed by private keys from the wallet. Nodes are responsible for adding blocks to their chain per their transaction bundling algorithm.

Because nodes operate both as a single-store and self-validators, off-node layer 0 backups are strongly recommended.

Nodes add one additional capability, a cross-chain reference —the ability to read or create a new transaction referring to a block on another node's chain. These references enable applications to interact with a definable subset of the system through a single node. The reference can optionally be direct with the node (peer-to-peer) or a hosted backup.

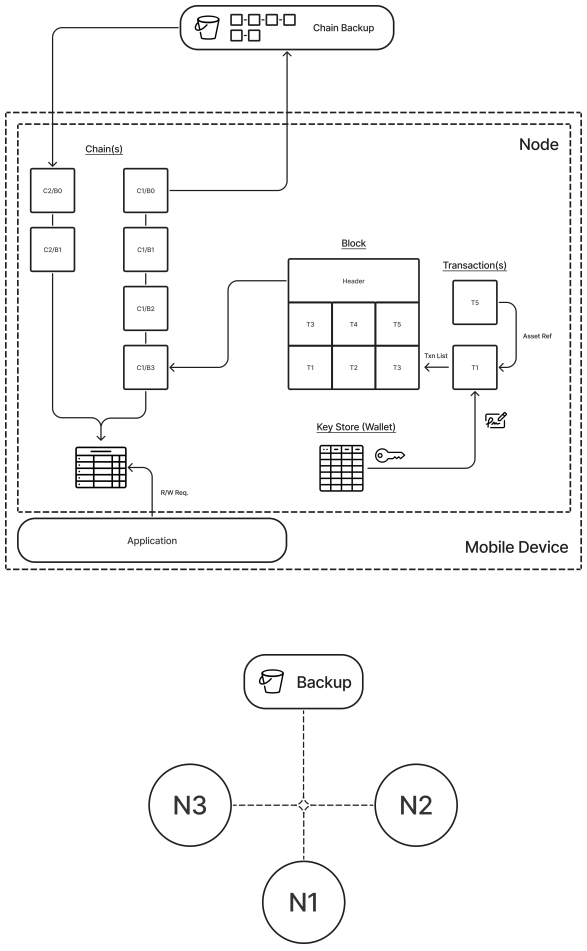

Fig. 1 Diagram of node structure and network composition

III. Transactions

There are two types of transactions for on-chain assets: initial tokenization and relative transactions. In

web3-lingo, initial tokenization is often referred to as minting. Where relative transactions relate to, directly referencing the initial tokenization. Examples of relative transactions could be provisioning access, ownership transference, asset modification, or even destruction.

A. Transaction Schema

Version (uint8) – a number indicating the set of transaction validation rules to follow

Addresses (uint8[32]) – an SHA-3 hash of the public key used for block signature.

Note: single address chain implementations may opt not to include transaction-level addresses.

Timestamp (uint32) – transaction creation time represented in seconds since epoch (Unix time[2])

Asset Ref – URI pointer to the original asset mint transaction. Or 0x00 for mint transactions.

Example: chain_id://address/transaction_id

Where transaction_id is the SHA-3 hash of the original mint transaction

Signature – an asymmetric digital signature (such as RSA or ECDSA) for the entire transaction, including the contents.

Note: RSA outperforms ECDSA in signature generation and verification for small transactions[1].

Contents – a binary encoded transaction payload. There is no max contents size, but contents are encouraged to stay under 100kB for performance.

Format:
[content schema byte length][content schema ID][payload]

**Schema ID must be <255 bytes in length*

B. Transaction Content Schema

The basic content schema is Binary Data, with the intention that implementations will add schemas for their respective applications.

Binary Data – Generic binary data. ID [0] with a variable length payload.

Example:
"hello world" (UTF8) = 010068656c6c6f20776f726c64h

IV. BLOCKS

Blocks follow the typical blockchain pattern of single child, single parent. Each block header contains a hash of the parent block header, with the block body containing a transaction count and transaction list.

While there is no max block size, blocks should be kept under 1MB for performance.

With the expectation of client-node deployment, transaction bundling and block creation are recommended once a minute. Transactions are serialized one after another in a list, with Transaction IDs (SHA-3) used to compose a Merkle root. Blocks are serialized and added to the chain.

A. Block Header Schema

Version (uint8) – a number indicating the set of block validation rules to follow

Timestamp (uint32) – transaction bundling completion represented in seconds since epoch (Unix time[2]).

Previous Block Header Hash (uint8[32]) – a SHA-3 hash of the previous block's header.

Transaction Root (uint8[32]) – the Merkle[3] root derived from all the SHA-3 hashes of all the transactions in the block.

B. Block Body Schema

Transaction Count (uint16) – the total number of transactions bundled in the block.

Transaction List – every transaction in the block, binary encoded, one after another.

Format:
[TX_1 byte length][TX_1 bytes]… [TX_n byte length][TX_n bytes]

V. VALIDATION

In addition to regular backups, periodic spot-checks are recommended to safeguard against node-level attacks. For example, run a randomized Acceptable Quality Limit (AQL) or flat 10% sampling of blocks once a day, performing both block and transaction validation. Irregularities can be restored from backup.

If restoring from backup, perform both block and transaction validation on each restored block.

A. *Block Validation*

Basic block-level validation requires a child-parent pair, hashing the parent block header and comparing for equality with the Previous Block Header Hash. This technique can be repeated until the entire chain is validated at a block level. Spot-checks should check up-chain at least five sequential child-parent pairs.

Dependent on the implementation, additional block validation rules may be added, such as max size or sequential timestamps.

B. *Transaction Validation*

Two different transaction validations are used, depending on the validation purpose.

For example, when restoring from backup, each transaction's signature is validated when performing a full validation. Also, the Transaction IDs are hashed and compared for equality against the block header's Transaction Root.

When creating a relative transaction (a transaction with a non-0x00 Asset Ref), the referenced transaction is validated using a Merkle proof. Merkle proofs may also be used to improve the performance of routine spot checks.

VI. Cross-chain Reference

Any relative transaction can reference an asset minted on another chain. Doing so requires a uniform resource identifier (URI) with an implementation unique chain identifier. Implementers may elect to refer directly to node-hosted ledgers or a backup —for example, *https://chain-id/address/block-id/transaction-id*. Single address chain implementations may opt to use only the address, chain id.

Nodes can be configured to cache blocks and transactions in a read-only capacity, using a local mapping table to route application requests to parallel ledgers. In addition to ad-hoc caching, some nodes may elect to implement periodic syncing. A common use case is an application running on both a mobile phone and tablet, creating transactions referring to the same asset.

VII. Tradeoffs

Everything has a tradeoff. The primary tradeoff in this proposed system is a larger burden placed on the implementor.

For example, a crucial design parameter is device security, particularly for private key storage. Modern device security systems provide more than adequate protection when configured correctly, but the responsibility to do so remains up to the implementor.

Implementors are also responsible for maintaining a list or map of valid addresses per their application. Otherwise, depending on the implementation, nodes could recognize invalid relative transactions. For example, address X mints asset Y, but then address X* creates a relative transaction stating the destruction of asset Y*. The issue is easily resolved if the application acknowledges that address X is valid and address X* is not. For most applications, addresses will be mapped to usernames resolving the issue instantly.

Akin to time-to-finality in shared ledger systems, without a shared ledger, nodes are vulnerable until blocks are backed up. For sensitive applications, nodes can be configured to disallow transaction interactions until successfully backed up.

The method to resolve time-conflicting blocks is at the implementors discretion and expected to be use-case driven. Simple solutions may range from chronological with exact time conflicts resolved by block size ordering to more complex on-chain locks.

While nodes can directly publish their ledgers for a peer-to-peer system, it is far more efficient to leverage centralized or semi-centralized backup storage for most applications. If backup centralization concerns implementors, nodes can replicate backups to multiple destinations. Alternatively, a decentralized autonomous organization (DAO) can be formed for decentralized governance over a shared asset.

VIII. Layer 0 Backup

Chain backup is per implementation; however, a public shared immutable repository is proposed. Each block is tagged with a backup epoch timestamp (Unix time[2]) and a node signature when backed up. Blocks are organized within the repository by *chain-id/address/block-id*.

In single address chain implantations, node signatures can use the same transaction signature private keys. Public keys (both node and transaction

level) are published to the backup enabling any nodes to perform validation as necessary.

A signed metadata file containing the last backup date and a list of recent blocks can be added to improve cross-chain syncing performance.

A potential cost-effective solution is to use a write-once, ready-many (WORM) repository commonly found in object storage solutions. Examples are Amazon Web Services S3[4] and Wasabi Object Storage[5]. Though implementors may opt for a more Web3 solution such as Arweave[6].

IX. Results

A proof-of-concept (POC) network has been deployed to stress test the system. The network consists of over 1,000 nodes that have minted over 3,500,000 assets in one month. Nodes routinely benchmark at over 10,000 transactions per second with peaks over 25,000.

The POC utilizes a node[7] written in Dart and packaged within a Flutter mobile application. Backup[8] is hosted in a public repository on Amazon Web Services S3 with Object Lock.

The POC implementation utilizes single address chains with one transaction per block resulting in an average backup cost of $0.00000425 per block per month.

X. Future Improvements

As the development of a complete system progresses, this paper will be appended with corrections and improvements.

Holochain[9] and their alternative approach were discovered during research into node-centric distributed ledgers —research into the implementation and lessons learned by the Holo team should be performed.

Examples or patterns for common implementor use-cases can be appended to this paper simplifying deployment. For example, resolution of multiple addresses or chains to a single application identifier.

For applications with multi-chain shared asset interaction, a defined pattern or structure would simplify implementors.

Finally, adding a Layer 2 or integration with a parallel Layer 1 blockchain protocol could add significant value. When considering additional asset ownership applications and use cases, many high-value, lower-frequency features exist to expand on. Some examples are asset sale, smart contract licensing, and pooling.

References

[1] S. Levy, “Performance and security of ECDSA,” 2015
[2] Wikipedia Contributors, “Unix time,” *wikipedia.org/wiki/Unix_time*, 2020
[3] R. Merkle, “Method of providing digital signatures,” *US patent 4309569 assigned to The Board of Trustees of the Leland Stanford Junior University*, January 1982
[4] Amazon Web Services, inc. “Amazon Simple Storage Service,” *aws.amazon.com/s3/*, 2022
[5] Wasabi Technologies, inc. “Wasabi Object Storage,” *wasabi.com*, 2022
[6] Minimum Spanning Technologies Limited, “Arweave protocol,” *arweave.org,* 2020
[7] Tiki inc., “localchain,” *github.com/tiki/localchain*, 2022
[8] Tiki inc., “syncchain,” *github.com/tiki/syncchain*, 2022
[9] Brock, et al. “Holochain – A framework for distributed applications,” *US patent 10951697 assigned to Holo Limited*, June 2020