Locus Chain Tech Whitepaper

Dec 1st 2021

Table of Contents

1. IMPORTANT NOTICE

Please read this section carefully. If you have any questions about what actions must be taken, please discuss with your legal, financial, tax, or other appropriate professional advisor(s).

Information in this whitepaper is subject to change or be updated and should not be construed as a commitment, promise or guarantee by LOCUS CHAIN or any other individual or organisation mentioned in this white paper relating to the future availability of services related to the use of the tokens or to the future of their value or performance.

This whitepaper does not constitute an offer or solicitation to sell shares or securities. It does not constitute or form part of and should not be construed as any offer for sale or subscription of or any invitation to buy or subscribe for any securities not should it or any part of it form the basis of or be relied upon in any connection with any contract or commitment whatsoever. LOCUS CHAIN expressly disclaims any and all responsibility for any direct or consequential loss or damage of any kind whatsoever arising directly or indirectly from reliance on any information contained in the whitepaper, any error, omission or inaccuracy in any such information or any action resulting therefrom

This is not a recommendation to buy or financial advice, and is strictly informational. Do not trade or invest in any tokens, companies or entities based solely upon this information. Any investment involves substantial risks, including, but not limited to, price volatility, inadequate liquidity, and the potential to complete loss of principal. Investors should conduct their own due diligence, with assistance from professional financial, legal and tax experts, on topics discussed in this document prior to making any investment decision.

All information herein are prepared from sources that are believed to be accurate and reliable. All market prices, data and other information are not warranted as to completeness or accuracy, are based upon selected public market data, reflect prevailing conditions, and our view as of this date, all of which are accordingly subject to change without notice.

The information contained in this document may include, or incorporate by reference, forward-looking statements, which would include any statements that are not statements of historical fact. No representations or warranties are made as to the accuracy of such forward-looking statements. Any projections, forecasts and estimates contained in this document are necessarily speculative in nature and are based upon certain assumptions. These forward-looking statements may turn out to be wrong and can be affected by inaccurate assumptions or by known or unknown risks, uncertainties and other factors, most of which are beyond control. It can be expected that some or all of such forward-looking assumptions will not materialize or will vary significantly from actual results

2. Introduction

Public distributed ledger systems are an innovative way to ensure trust in the information. Until distributed ledger systems emerged, centralized information systems were the only way to ensure trust in the stored information, but only within the trust of the system managers.

The public decentralized distributed ledgers operate on the distributed P2P networks, free from manipulation by malicious participants.

Decentralized distributed ledger technology allows people to grant trust in information stored in P2P networks. With distributed ledger technologies, a.k.a. "Blockchains," it is now possible to establish objective credibility on information publicly shared.

The decentralized distributed ledger technologies (or Blockchain platforms) guarantee trust by ensuring aspects like immutability, traceability, and openness to information.

In a blockchain, information is immutable because no one can randomly forge or manipulate the information recorded on the ledger. The information is signed and timestamped. Information is traceable because each chunk of information is strictly tied to other information. Blockchains are open because the networks are operated by virtually anyone that participates in the public blockchain networks.

However, the current distributed ledger technologies have technical issues that must be solved. One of the critical issues is the performance of a Blockchain system. Simply, we need to achieve a goal of practical volume and speed of transactions in a large-scale blockchain network. Nevertheless, this is not a trivial problem.

A typical solution to the performance problem is allowing centralization to the blockchain system. This approach is criticized for heavily damaging the intrinsic value of decentralization.

Locus Chain is a distributed blockchain system with the goal of processing at least 4K transactions per second on a decentralized network. The Locus Chain network will consist of various devices, including generic home PCs, data center servers, mobile phones, or low-power IoT devices, all in different ranges of CPU performance, storage capacity, and network speed. The Locus Chain also has a goal of transaction processing time, reaching an immediate consensus within a few seconds, and reaching a deterministic permanent consensus within a few minutes.

Locus Chain proposed and implemented several technological breakthroughs to reach these goals.

For example, Locus Chain enables parallel transaction processing by incorporating a nonlinear DAG structure called AWTC (Account-Wise Transaction Chain) instead of a linear "chain" structure.

Also, Locus Chain implemented a PoS-based BFT consensus algorithm to resolve the shortcomings of PoW and Nakamoto Consensus regarding performance and determinability while keeping fairness between participating nodes. Furthermore, Locus Chain implements the Dynamic Sharding data-distribution policy to decrease network loads while keeping the ledger's integrity.

Locus Chain's Verifiable Pruning technology dramatically reduces the storage space requirement of nodes, and small-footprint computers like IoT devices and mobile phones are eligible to be a part of the Locus Chain network.

Furthermore, Locus Chain is aware of the risk of the emerging Quantum computing technology to blockchain systems. Locus Chain implements a post-quantum resistant cryptography system by separately managing the master key and the regular key.

3. Basic Technological Concept and Locus Chain's Ledger Structure (AWTC)

(1) Locus Chain's ledger

Locus Chain is a distributed data storing system in which the participators can freely add data. The sum of the saved data on Locus Chain is called the ledger.

In a narrow sense, ledger is the sum of data blocks (transactions) that are intentionally added by the participators. In a broad sense, ledger refers to all stored information required for Locus Chain's normal operation including the information generated during the consensus process.

(2) Account

An account is an actor that can actively add data on Locus Chain's ledger. An account is defined by its Secret Key/Public Key pair. Each Account could be uniquely identified by its Address, which is calculated from the public key. Account addresses are the preferred way of referring accounts in the Locus Chain system. Besides, an account uses its secret key to sign the operations performed on the Locus Chain system. For example, when an account issues a transaction to add some block of data to the ledger, the block must be signed with the account's secret key to prove its legit ownership.

(3) Transaction and Account-Wise-Transaction-Chain (AWTC)

The AWTC, Account-Wise-Transaction-Chain, is a central data structure for Locus Chain's high-capacity, distributed transaction processing.

AWTC is a Directed-Acyclic-Graph (DAG) based data structure composed of multiple transaction chains, each for an account. Each account has its dedicated chain. A new transaction issued by an account is primarily added to the account's chain.

Typical linear ledgers have only one spot for a new block to be connected to the previous block, which may become a bottleneck for high-speed data processing. In contrast, AWTC's DAG-based structure has multiple parallel data insertion points at any time, and there is virtually no limit of transactions added at a time. Also, collisions do not occur because a new transaction is appended to its own single fixed point on the AWTC graph and settled immediately if the issuer account is not malicious. This characteristic fundamentally resolves the transaction processing delay issue of conventional blockchains.

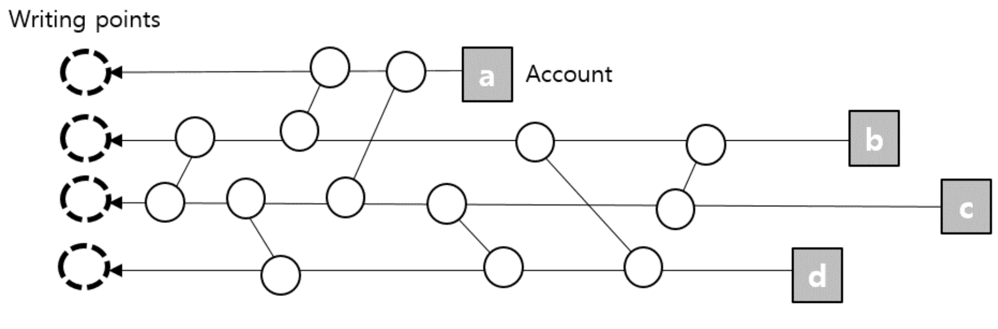

Figure 1 AWTC structure

A Transaction (Tx) is a unit of data added to the ledger. When an account wants to add information to Locus Chain's ledger, the account issues a transaction containing the information. Each transaction is issued solely by a single account. The issuer account signs the transaction with its secret key.

There are two ways to identify a transaction; hash value and index number. A Transaction Hash Value is a calculated result of a cryptographic hash value of a transaction's entire data. A hash value is suitable for pointing to an arbitrary transaction in the ledger.

A Transaction Index Number is a unique, sequentially increasing id number for transactions of a specific account. The first transaction of the account's AWTC is a Tx with index number 0(zero). Each consequent transaction has an index number increased by one from the previous transaction. By this, a transaction could be identified by the account address and index number.

The index number clearly specifies the order of transactions of an account. A set of transactions of an account, with its order preserved, is called a Transaction Chain. An account cannot delete or modify the previously issued transactions. The account can only 'extend' the chain by issuing a new transaction.

Each account has only one transaction chain, and Locus Chain's ledger is equivalent to the sum of all account's transaction chains.

An account cannot manipulate another account's chain. However, the account may issue a transaction regarding requests to other accounts to its Transaction Chain. Such transaction acts as a message from an issuer account to other destination accounts. When a message transaction is propagated and reaches the destination accounts, then the destination accounts may issue transactions referencing the message transaction.

Let us see an example. Assume that account A wants to transfer some coins to account B. First, account A generates a transaction like {A: -10 from self, +10 to B}, adds to A's chain, then broadcasts it over the network. At this point, A's coin is reduced by 10, but B's coin is not changed. When B observes the A's transaction, then B completes the transfer by issuing a transaction like

{B: +10 received from A} referencing the A's transaction. Now the B's coin is increased by 10, and the transfer completes.

The first transaction (0th transaction) of each account means the new creation of the account. 0th transaction always occurs as a coin-receiving transaction from another account. All accounts except for the genesis system accounts are derived from other accounts.

In summary, AWTC is a fundamental basis for Locus Chain's high-capacity transaction processing.

(4) Network, Nodes, and Shards

Machines in Locus Chain communicate over the Internet. Each computer that participates in Locus Chain is called as a Locus Chain node (or just a node). Each node could communicate with other nodes, mainly transmits and receives data using Peer-to-Peer (P2P) style gossip network. The Locus Chain network is the total sum of nodes that carry out the communication for Locus Chain. In the case of Bitcoin and Ethereum that use PoW and Nakamoto consensus, massive computing power is required to run a node. However, Locus Chain has been designed to run not only on household PCs but also on low speculation devices such as mobile devices.

A Shard refers to a portion of Locus Chain's network that divided into an adequately sized group of nodes. A node can only participate in one shard at a time. A node may communicate with any nodes in any shards, but it must prioritize communication with nodes in the participating shard.

From the view of ledger management, each shard manages a portion of the entire AWTC account chains. In practice, each shard manages Transaction Chains that are managed by all nodes in the shard. A node may access the entire ledger data, but the node should sync and duplicate AWTC account chains of the participating shard in a priority. The consensus algorithm also runs in the per-shard basis since each shard manages a different subset of whole ledger data.

4. Consensus Algorithm

The AWTC structure, a form of DAG used in Locus Chain, is designed to settle a transaction in a "retail processing" time scale, usually within 10 seconds. This is possible because collisions between accounts do not occur, and the result is unlikely to change in most cases. However, an account owner may cause problems like double spending on its own Transaction Chain if they are malicious. To handle situations like this, Locus Chain executes a kind of BFT consensus periodically in every few minutes, to check and resolve problems.

Double spending in the AWTC structure is defined as a case of two or more transactions with the same index number are simultaneously issued to an account's transaction chain.

Even if multiple different transactions with the same index number are issued simultaneously, the transactions propagate over the network through different nodes. During the propagation, the sum of (delegated) stakes over propagated node paths of each transaction could be considered as a pseudo-voting count. The transaction with more votes is decided as the primary candidate of a nondeterministic consensus. Then, the BFT consensus committee nodes, who are elected in a stochastic dPoS manner, periodically (for roughly every 2 minutes) confirm and finalize the consensus in deterministic form by signing the ledger status.

Each node updates its view of ledger status independently because AWTC is a parallel data structure. Thus, a node cannot know the absolute state of the whole ledger in a hard real-time manner. Locus Chain executes a deterministic consensus algorithm against slightly old status of the ledger, offsetting the average time required for a transaction to propagate within the shard.

Locus Chain's consensus algorithm has improvements over the traditional stop-and-go method because our algorithm does not disrupt issuing transactions during the consensus algorithm and also does not interrupt AWTC's high-speed processing.

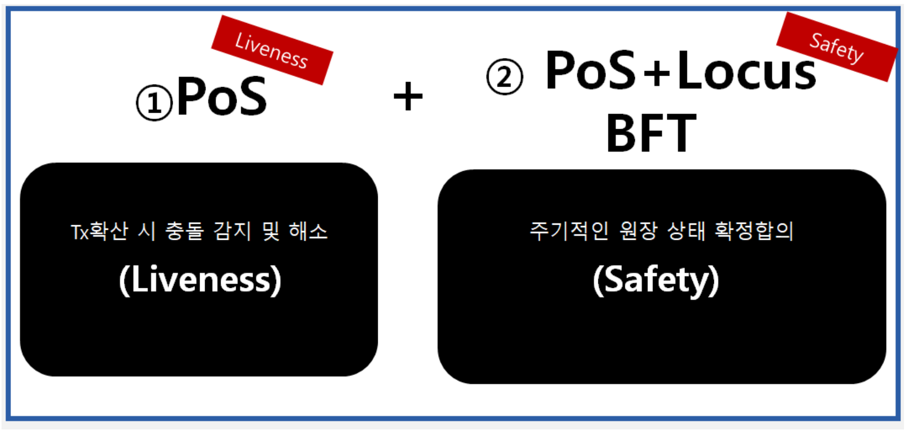

Figure 2 Collision Resolution and Consensus Algorithm

The BFT deterministic consensus is implemented in the AWTC ledger structure as mentioned above and is called the Locus BFT consensus. It is meaningful to technologically implement DAG and BFT simultaneously since they become the basis of pruning and sharding technology to resolve the storage and network load problem later on. Unlike other projects, Locus Chain accurately implemented BFT on the DAG structure, which has been advanced to resolve the storage and network load issue simultaneously.

(1) The Core Idea of Locus Chain Consensus Algorithm

A blockchain system should have a ‘consensus algorithm,’ which is an algorithmic method to ensure the reliability and integrity of the proposed information, over whole participants, in an open manner.

Based on Locus Chain’s Account-Wise Transaction Chain (AWTC), which is essentially a Directed-Acyclic-Graph structure, every transaction in Locus Chain is explicitly ordered and grouped by its issuer account. In AWTC, the transactions can be easily verified using its issue order and issuer account. Soon, every node in the network group eventually receives the same transaction and shares it according to the characteristics of the gossip network.

By this, we can assume this; “After a reasonable amount of time, the majority of network nodes share the same received transactions. Then the majority of honest network nodes may reach an agreement on the contents of received transactions.”

Locus Chain implements this idea by grouping each transaction into *Rounds*, which are separated periods of appropriate time intervals. Then transactions included in each Round are identified, ordered, and digested into a hash value. Then the consensus algorithm is executed for each Round to agree on an identified hash value.

(2) Building blocks of Locus Chain's Consensus Algorithm

We explain the concepts of basic components for Locus Chain consensus algorithm below.

Shards / Nodes

As described previously on shards and network nodes, transactions of an account are primarily managed by the shard that the account belongs to. A consensus algorithm is also executed by the shard, which is a unit of network nodes that receives and shares the transaction data.

Each shard is composed of roughly the same number of nodes. An important limiting factor of the size of a shard is the average time required to propagate a transaction over a shard fully. The number of nodes should be small enough to guarantee transactions to be delivered to most of the nodes within the same shard in a reasonable time.

Locus Chain assumes that shards and nodes have relatively accurate internal clocks. The clocks are accurate within a few seconds. Each node should be able to measure the start and end of a round and communication delay to a certain extent.

Transactions

A transaction is a signed chunk of a data unit to be added to the Locus Chain ledger. Transactions are signed with the issuer account's secret key. The sign could be checked against its issuer's public key to verify the identity of the issuer.

An account must set a unique integer serial index number to all the issued transactions. The first transaction of an account must have the index number of 0, and subsequent transactions by the same account must have index numbers increased one by one. Also, each transaction (excluding the first transaction) must include the hash value of the transaction of the previous index number. Also, a new transaction must include the round number in which the transaction should belong.

The order of transactions of a specific account is verified using the index number and the hash value of the preceding transaction. Each node verifies the validity of transactions upon receiving using the transaction's index number and the hash value of the previous transaction. Nodes participating in the gossip network and the consensus process must only use verified transactions. Accordingly, nodes should keep a reasonable amount piece of information required for verifying transactions.

If a node receives an invalid transaction, then the transaction is discarded by the node. For example, when a node receives a transaction that contains an invalid round number, the node automatically ignores the transaction.

Rounds

A Round is a basic time unit of collecting and processing new transactions. Rounds are set to around 2 minutes long currently. Rounds have *Round Number*, an integer index number that increases by 1 for each round. The transactions having the current round number are processed on the next consensus process. The consensus process runs for a round is executed in the middle of the next round considering the network propagation delays of transactions.

Round State Proposer Committee and Voting Committee

Locus Chain's consensus algorithm executes on per-round basis. The consensus process is executed by the limited number of nodes that randomly selected in a fair manner for each round. The process uses predictable and relatively small footprint of computing power and communication cost because the algorithm runs on the limited number of selected nodes.

Elected nodes form committees. Two committees (two groups of nodes) elected for each round. One is group of *Round State Proposer Committee*" and the other one group is "Round State Voting Committee."

"Round State Proposer Committee" is a group of nodes ("RS proposers") capable of proposing a Round State. When the consensus algorithm starts, each proposer (node)generates a hash digest value of transactions received for the round, then signs the hash value with its own key, then broadcasts the signed hash value as a *Round State Value Candidate*.

"Round State Voting Committee" is a group of nodes ("RS voters") capable of verifying Round State Value Candidates. Each RS voters verify and compare the proposed RS value candidates, then vote for a value closest to the voter's own received transactions.

(3) Steps of Locus Chain's Consensus Algorithm

Locus Chain's consensus algorithm runs through the following steps.

Step 1: Committee Election. In the middle of a round, each shard randomly elects RS proposers and RS voters for the round. Currently, the number of proposers is adjusted to about 5, and the number of voters is adjusted to about 50.

Step 2: Round State Proposal. After a certain time passed from the end of the round, each node of RS proposers calculates a hash value based on the received transactions, then proposes and submits the hash value as a Round State value candidate.

Step 3: Round State Voting. Each node of RS voters verifies the submitted Round State Value Candidates from step 2, then cast a vote for the value that is the closest to the node's transactions.

Step 4: Round State Confirmation. Each node of RS voters receives the votes from step 3, then finds a Round State value candidate that gains enough votes, typically at least 2/3 of all votes. Then each RS voters sign the found value for confirmation and explicitly broadcasts the result. The signed results of each node of RS voters are permanently recorded to be included in the round and could be consensus proofs and activity records of the voting committee.

(4) Validity of Transactions and Locus Chain's Consensus Algorithm

In Locus Chain, the validity of each transaction could be verified without the Consensus Algorithm in most cases. Because of the nature of Locus Chain's AWTC ledger structure, only the account owner (who possesses the account's secret key) can issue and add transactions to the account's transaction chain. Every transaction added is a proper and valid transaction, unless the account with the secret key commits wrongful acts on purpose, such as the Byzantine behavior.

Byzantine accounts may create double-spending situations by issuing two or more different transactions (double-spending transactions) with the same index number. However, the second transaction is dropped by network nodes if the first transaction propagates properly before the second transaction appears. For example, our design assumes that about 10 seconds is enough for propagating a transaction over an entire shard. If the time interval between two double-spending transactions is long enough to propagate the first transaction over the whole shard (say, about 10 seconds), or double-spending transactions are not issued in the same round, the second transaction is dropped by each network node, and only the first one becomes valid.

The problem becomes complicated if two or more double-spending transactions are issued simultaneously. It is hard (or not feasible) to decide one transaction uniquely as the valid one without executing the consensus algorithm. However, each node may detect simultaneously issued transactions by comparing the receiving timestamps of double-spending transactions. If a node received two or more transactions of the same index number in a relatively short period (say again, 10 seconds), then the node may mark the transactions as unfaithful and stop trusting those transactions until the consensus algorithm resolves the conflict.

In summary, transactions are valid and unforgeable even without the consensus algorithm, unless the secret key owner explicitly conducts Byzantine double-spending activities. Locus Chain's consensus algorithm resolves the key owner's malicious double-spending transactions if exist. Participating nodes can detect and mark malicious transactions within a few seconds in most cases. Finally, the consensus algorithm resolves conflicting transactions and builds additional stability for Locus Chain Network operation, including Sharding and Pruning.

(5) Detailed Steps of Locus Chain's Consensus Algorithm

The core idea of Locus Chain's Consensus Algorithm is based on the property of a gossip network, which distributes information over network nodes fairly and repeatedly. This property causes all network nodes to receive the same information eventually. Our algorithm is about finding out the information received by the majority of network nodes.

Our consensus algorithm can be roughly described as follows; First, the state of the received information is proposed by nodes as several digested hash numbers. Then the proposed hash numbers are counted and compared, then one of the hash numbers is selected as the official state of received information.

Our algorithm executes on a per-round basis. The algorithm consists of four steps; "Committee Election Step," "Round State Propose Step," "Round State Selection Vote Step," "Round State Commitment Step."

The role of each step is as follows.

- **Step 1: Committee Election Step**

Not all nodes in a shard participate in the consensus algorithm. The algorithm is supposed to be processed by a certain number of nodes, to balance and optimize the network and computational resources.

Two groups of nodes (i.e., two committees) are needed to run the algorithm. The first group of nodes is '(Round State) Proposer Committee,' which proposes Round State Values. A Round State Value is a digested, representative hash value of received transactions during a round. The second group is '(Round State) Voter Committee,' which verifies, counts, and votes for the proposed Round State Values. Both committees are newly elected for each round.

Committee member nodes are elected using reproducible pseudo-random numbers that can be reproducibly calculated using disclosed information. For example, A Verifiable Random Function (VRF) using disclosed ledger information is a good fit for this purpose.

When a new round begins, each node in a shard checks whether it is selected as a committee member or not, by calculating its own pseudo-random value for the round. Each of the elected nodes broadcasts the 'Election Message,' which contains the node's identity and information that can be used to verify the calculated random number. All nodes in the shard listen for the Election Messages to find out the number of committee members that have been elected.

Each round of a shard has its own desired size (i.e., number of nodes) of committee groups, but the independent random calculation of the election process makes the actual elected number different for each election. The actual number of nodes for committees can be figured out by gathering election messages. Most of the time, a sufficient number of nodes become elected within acceptable error range.

Elected committee nodes are rewarded with incentives if they faithfully participate in the consensus algorithm.

- **Step 2: Round State Proposal**

Each round has a fixed start time and end time. When certain time passes after a round's end, the Proposer Committee proposes Round State Values. Each node in the Proposer Committee independently calculates its Round State Value by digesting transactions received during the round.

Currently, Round State Value is the top-level hash value of a Merkle-Patricia Trie (MPT) composed of transactions during the round. Each proposer committee node creates an MPT by inserting transactions into the MPT. MPT is a kind of Radix Trie. Like other trees and tries, an MPT is generated by adding elements one by one. The insertion order does not affect the MPT. In other words, the same MPT will be generated regardless of the order of the inserted elements. Also, each trie-node of an MPT contains a representative hash value. Each proposer committee member node creates its MPT by adding all incoming transactions in real-time. The calculated final value of the MPT's root becomes the Round State Value for the committee member node.

After calculating the Round State Value, the committee member node broadcasts a Round State Proposal Message (or just Proposal Message) signed with its secret key. The message contains information including Round State Value and the number of transactions received.

- **Step 3: Round State Selection Voting**

After the Round State Proposal Step, the shard becomes flooded with Round State Proposal Messages as many as the number of Proposer Committee nodes. Round State Selection Voting is the process of selecting one proposed round state which mostly suits the round.

Each member node of the Voter Committee listens for Proposal Messages. If a Proposal Message of a certain condition is received, or if a sufficient number of Proposal Messages are received, the node selects a Proposal Message which suits the following predefined criteria.

  - ➢ Criteria 1: Select the proposal in which the Round State Value corresponds to the voter node's own Round State Value.

- Criteria 2: If criteria 1 is not satisfied, select a proposal with transaction count (number of transactions) closest to the node's count of received transactions.

- Criteria 3: If there are multiple proposals match for criteria 2, compare the numerical value of the Round State Value and the pseudo-random election value of the Proposer Node, to choose one of the matched proposals.

The voter node selects one proposal, then composes and broadcasts a Selection Voting Message signed with its secret key.

- **Step 4: Round State Commitment**

Each node of the Voter Committee listens and counts the Selection Voting Messages issued in the previous step.

If one single Round State Value collects over 2/3 of all the Selection Votes, the Round State Value is regarded as "confirmed" for the state of the round. When a Voter Committee node observes a Round State Value that has 2/3+1 votes out of the whole Voter Committee size, it marks the Round State Value as confirmed, and then composes and shares a "Round State Confirmation Message" for the confirmed Round State Value. The confirmed Round State Value has multiple signs of several voter nodes. Signing algorithms like Schnorr Signature are used to optimize the signing process.

All nodes in the shard listen for Confirmation Messages. A node may regard the round as confirmed when the node receives enough number of signs for a certain Confirmation Message.

Each node of the Voter Committee issues a "Confirmation Failure Message" (or Null Confirmation Message) when a Round State Value does not collect 2/3+1 of the votes. If a node receives a sufficient number of Failure Messages, then the round is considered to have failed to reach an agreement. The failed round is re-resolved by taking further actions.

(6) Round State Chain

Transactions of Locus Chain are placed in the AWTC structure and managed by its owner accounts. However, the result of the consensus algorithm does not belong to any particular account. Locus Chain uses a special type of global transaction chain, called "Round State Chain," to manage the information delivered by the consensus process.

The Round State Chain contains global information for each round, including the MPT hash value of the ledger state, number of included transactions, information regarding Proposer Committee and Voter Committee, and hash value of the previous round state. In reality, this information is already contained in the confirmed round state and stored almost as-is on the Round State Chain.

(7) Clock Timing of the Consensus Algorithm

Each step of Locus Chain's consensus process executes in sync with certain time schedules. Currently, each round of Locus Chain is about 2 minutes long, which means all steps should be completed in two minutes. Starting time and timeout of each step are decided accordingly. Also, the size of shards and committees are adjusted regarding performance goals.

The first step ("committee election step") begins simultaneously with the round. The second step is delayed to a certain time after the end of the round, to ensure the propagation of transactions over the shard. Currently, the delay is around one-minute long.

Because the clock timing is critical for our algorithm, we assume that each network node has a clock that is accurate enough with a maximum of several seconds of error. If required, a clock-matching protocol is used to detect clock errors.

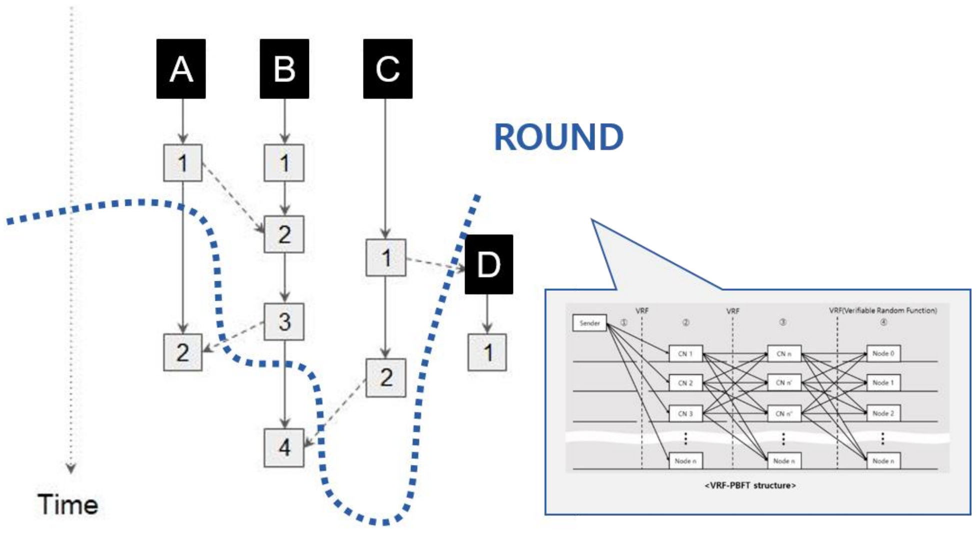

Figure 3 Locus BFT Consensus Algorithm

(8) Detecting and Rejecting Invalid Transactions

The consensus algorithm of Locus Chain is based on an idea about how to decide a set of transactions that have properly propagated to certain number of nodes within a certain time interval. We assume that transactions are issued properly by honest accounts and nodes, but also assume that there could exist 'invalid' transactions for some reason.

An 'Invalid transaction' is a transaction in which contains information contradicts to (or conflicts with) the existing confirmed transactions and other information. For example, a transaction with a sign that cannot be verified with the known account public key is an invalid transaction. Or, a transaction with the transaction index that does not follow the confirmed previous transactions is also an invalid transaction.

Such invalid transactions are ignored by nodes to prevent further propagation. Every node in a shard must validate received transactions to detect invalid transactions.

There are cases in which a newly received transaction does not conflict with previously confirmed transactions, but conflicts with other transactions in the same round. For example, if an account issues two transactions with the same transaction index, each transaction may not conflicts with the previously confirmed rounds, but the two transactions conflict with each other. If these conflicting transactions sent separately to different nodes, the receiving nodes could not reject the received transaction because the transaction does not individually violate the previous rounds. In this situation, the whole shard may contain conflicting transactions, which may lead to a problematic 'double-spending' situation. However, there are chances that each node may resolve conflicts through predefined rules when both transactions arrive, usually before the end of the round.

(9) Cases of Consensus Failure

If most of the nodes are honest and the network connection is stable, it is quite unlikely that a consensus process fails. However, failure can occur in the following cases.

- **There was no round state proposal in Step 2**

In this case, a timeout will occur on step 2 and step 3. Consequently, step 4 will reach an agreement of "consensus failure[2]" with more than 2/3 votes of "Confirmation Failure Messages."

- **The Round State Selection Votes did not gain over 2/3 votes in Step 3**

In this case, a timeout will occur on step 3. Consequently, step 4 will reach an agreement of "consensus failure" with more than 2/3 votes of "Confirmation Failure Messages."

- **The Round State Confirmation or Confirmation Failure Votes did not gain over 2/3 votes in Step 4**

In this case, a timeout will occur on step 4, which leaves the consensus result "undecided." This case is the most problematic. A critical network failure may prevent proper communication in a shard, and there could be inconsistencies in the consensus result. In other words, some major numbers of nodes may not receive the proper result of the consensus at the end of the round, leaving the round undecided for the majority of nodes.

(10) Retrying Consensus on Failure

When a Round Consensus is reached with major votes for Confirmation Failure Messages in Step 4, or when the consensus left undecided by a timeout in Step 4, each node must regard the round as 'failed' and should start a re-consensus process, which is effectively a process of retrying a Round Consensus.

The consensus algorithm for retrying a Round Consensus is the same as the normal consensus algorithm of Locus Chain, but with parameters changed. All transactions of failed rounds are included in the re-agreement process. At this time, transactions would be more promptly propagated over the shard because a certain time has passed since the end of the round. Each node may re-sync the received transactions with other nodes.

The number of nodes in committees for the re-agreement process are increased from the normal consensus process. In the case of two or more confirmed round states are later emerges, the total stake of nodes participated in the committee would be a good criterion for deciding most suitable state. If a node sees multiple different confirmed round states, the one with the largest total stake must be selected as the major state. (This special case may occur when a round confirmation result is not propagated from some network failure, and then re-broadcasted after network recovery.)

5. Economic Structure (Incentives, Coin, Gas)

(1) Incentives for Locus Chain's Participators

Voluntary participation of Locus Chain's accounts and nodes are essential to keep the Locus Chain system running. Locus Chain has the concept of Coin and Gas, as well as other blockchain systems to incentivize the participating nodes and accounts. While any data could be added to the Locus Chain system, the Coin and Gas are special baked-in system values required for Locus Chain.

(2) Coin and Gas

Coins are numbers which represent general measure values in Locus Chain. Coins can be transferred between accounts. An account must receive some amount of Coins when created. In other words, the first transaction of every account must be a Coin-receiving transaction (except for the accounts created in the genesis round).

Gases are numbers that represent system-internal values for operating the Locus Chain system. For example, an account must pay Gas for creating a transaction. Additional Gas may be required if the transaction contains additional services like smart contracts. Coin may be consumed ("burned") instead if the account does not have enough Gas.

The amount of Coins and Gases are publicly disclosed information. Any participant can calculate the amount of Coins and Gases of every account from the transaction history.

(3) Coin Stake

An important role of Coins is valuing the influence (or importance) of an Account. The amount of Coins (or Stake) of each account is used as a weight-value for electing the Committee members for each round. Locus Chain forms the delegated-Proof-of-Stake(dPoS) scheme, which means accounts with more stake have higher chance to be become elected.

An account may delegate its stake to another account. Accounts in the offline state may delegate its stake to another active online node to utilize the stake. A node's stake is the sum of its primary account and all delegatee accounts.

(4) Epoch: Calculating Incentives

Coin is an incentive to nodes and accounts that contribute to operating the Locus Chain system. Coin incentives are calculated based on the history of transactions and round-states, using the pre-fixed formula. The result will be the same for any node that calculates the incentive for each round.

Transactions are fixed and agreed on a per-round basis. Incentives are calculated for several rounds for efficiency. Currently, Locus Chain calculates incentives for a bunch of rounds for one whole day. The round that calculates the incentives is called an 'Epoch.' In other words, incentives are calculated and generated in per-epoch basis.

A round used as a basis of incentive calculation is called an 'epoch pivot round.' Normally an epoch pivot round points to several hours before the epoch-changing round. Each node in a shard executes the calculation using the ledger information at the epoch pivot round. The calculation delivers incentive Coin and Gas numbers, and reference total-stake and reference node count, which are important parameters for the round consensus. The reference of total-stake and node count value are fixed constant values that are effective until the next epoch.

The values such as the calculated incentives on the epoch standard round and others are executed along with the consensus of other transactions in the corresponding round and leaves the results on the ledger. The shard uses the newly calculated value in the consensus algorithm starting from the next round (epoch changing round) after being recorded on the ledger.

The calculated incentive values are submitted for the round consensus to reach an agreement. When agreed, the next round will be the round of an epoch-change in which the calculated numbers come into effect.

Each account eligible for receiving Coins and Gases must submit the incentive-receiving transactions to collect the incentives.

Gas as an incentive is automatically given to the corresponding account on a per-round basis. Additionally, for each epoch, each account's Gas may automatically become filled to a certain amount ('Free Gas Amount') if the account has less Gas than the minimum free Gas amount.

(5) Coin as an Incentive Value

Coin generated as an incentive is an assessed value of the contribution of each Account in the Locus Chain System. Several factors are considered in evaluating the contribution. For example, the amount of Coins rewarded is designed to gradually reduced over time to incentivize the early participants. Also, the Coin Stake and the actually generated Coins is somewhat non-proportionately balanced to encourage new participants.

(6) Gas

Gas is the internal value required to issue a transaction. Gas currently cannot be purchased or given. Each account may be supplemented with some free gas for each epoch. Accounts that participated in a round consensus may receive additional Gas.

Gas enables micro-transactions by allowing accounts to submit transactions without consuming Coins. The limited amount of free Gas prevents transaction spamming. An account must use Coins to submit transactions when it does not have enough Gas.

Gas is also a compensation for contributors of the Round Consensus. Unlike the Coin, which requires epoch change, Gas is quickly generated on a per-round basis.

6. Verifiable Pruning

(1) Utilizing the Data Locality: Ledger Sharding and Verifiable Pruning

Shard Data Locality: Ledger Sharding

In the traditional sense of blockchain, the ledger is in the form of one single chain. Each node has to hold all ledger data to check the integrity of data, both the existing and the new.

However, the holding-all-ledger model is not suitable for sharded ledgers. In its nature, each independent shard is supposed to have zero knowledge regarding other shards. A node only verifies transactions included in the current shard. The shard consensus only handles transactions issued by accounts in the shard.

This property also implies that a node can safely remove (or "prune") the data regarding other shards. While communications like inter-shard transactions carry data about other shards to a node, the data is temporary and does not have to be kept in the local storage for a long time.

The ledger sharding effectively reduces the size of the local storage of each node by the number of shards.

Interest-based Data Locality

The integrities of data (like transactions and blocks) are checked against the ledger. In practice, the verification requires a (very) small part of ledger data. For example, each new data is verified only with the recently verified data on the chain in practice. Older data are there to keep the security chain, but they are mainly inactive once the integrity of the newest data is secured.

This property raises an interesting question; what happens if we could temporarily remove the old data? Locus Chain elaborates the questions to an idea of Verifiable Pruning. With Verifiable Pruning, A Locus Chain node can remove (or prune) most of the transactions and blocks which are out of interest from local storage.

Verifiable Pruning

If a node removes a data chunk and fetches it later again, the node must also verify the data again. In the sense of traditional blockchains, verifying a chunk of data requires all proceeding chunks of data in the chain, which requires a significant amount of computation.

Locus Chain's Verifiable Pruning is about efficiently verifying a chunk of data without scanning all previous data chunks. Verifiable Pruning uses Hierarchical Skewed Merkle Tree (H-SMT), a patented unique data structure to exponentially reduce the overhead of data verification. With Verifiable Pruning, verifying a data chunk requires only a small number of previous data chunks, around a number regarding *log_{10} (N)*. The small number of required data enables verifying all required chunks in less than a second.

For example, when a Locus Chain node recognizes an old transaction that was not in the interest, then the node can identify a set of other transactions required to verify the transaction. The set of transactions may be small enough to be fetched and verified before the next consensus.

Verifiable Pruning enables the saving of ledger storage space. Furthermore, Verifiable Pruning enables fast startups of nodes. In Locus Chain, a node uses only a small set of previous blocks to verify the integrity of the current head. A fresh new node could participate in a shard within a couple of consensus rounds.

Data Ownership and Responsibility

Verifiable Pruning enables pruning data of no interest. In other words, a node must not prune the data of interest. Typical non-prunable data are the transactions regarding the node owner account and dangling accounts.

Each node has an owner account. A node may have dangling accounts that are not operating a node. A node must keep track of data regarding relevant accounts and serve the data to other nodes if requested.

Pruning is an essential operation for all nodes. Especially low-capacity nodes such as IoT modules could take advantage of pruning. On the other hand, nodes with sufficient storage space may contribute to other nodes by acting as achiever nodes.

(2) Hierarchical Skewed Merkle Tree: A data structure for Verifiable Pruning

A Traditional Skewed Merkle Tree

Hierarchical Skewed Merkle Tree (H-SMT) is the central data structure that supports Verifiable Pruning of Locus Chain. H-SMT is based on Skewed Merkle Tree, which is a variant of Merkle Tree.

Merkle Tree is a tree data structure that each leaf node has a label of cryptographic hash value, and each non-leaf node has a label of cryptographic hash value delivered from its children's label, which are also hash values. In blockchains, Merkle Tree is widely used to prove the existence of data in a list.

Skewed Merkle Tree (SMT) is a form of Merkle Tree that each non-leaf node has one leaf node and one non-leaf node. Skewed Merkle Tree works as a kind of Linked List based on Merkle Tree. A data is added to SMT linked list by adding a new root node on top of the previous SMT. The previous root becomes a child to the new root node, and the new data block is also added as a child. The number of data blocks in an SMT is the height of the SMT minus one. A node in an SMT is verified by checking hash values in its children, effectively calculating the hash of all nodes preceding the node.

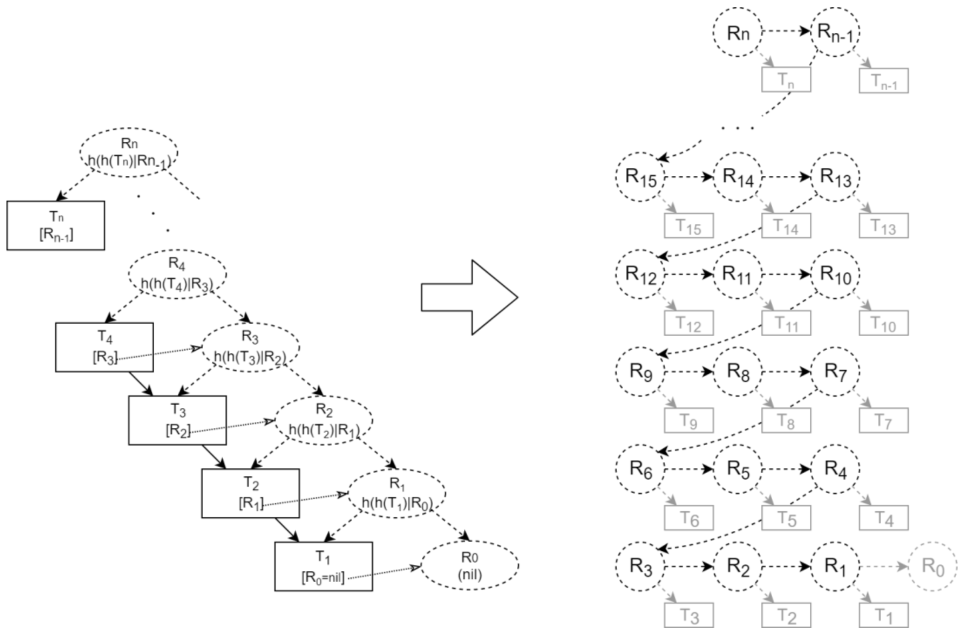
R_n $h(h(T_n)|R_{n-1})$
T_n $[R_{n-1}]$
R_4 $h(h(T_4)|R_3)$
T_4 $[R_3]$
R_3 $h(h(T_3)|R_2)$
T_3 $[R_2]$
R_2 $h(h(T_2)|R_1)$
T_2 $[R_1]$
R_1 $h(h(T_1)|R_0)$
T_1 $[R_0=nil]$
R_0 (nil)
R_n R_{n-1} T_n T_{n-1}
R_{15} R_{14} R_{13} T_{15} T_{14} T_{13}
R_{12} R_{11} R_{10} T_{12} T_{11} T_{10}
R_9 R_8 R_7 T_9 T_8 T_7
R_6 R_5 R_4 T_6 T_5 T_4
R_3 R_2 R_1 R_0 T_3 T_2 T_1

Figure 4 A Skewed Merkle Tree, and its simplified representation

Hierarchical SMT: A Directed Acyclic Graph built on SMT

An SMT is a straightforward data structure. The cost of verifying an SMT is proportional to the data length, or O(N). Hierarchical SMT improves the performance to O(log N), by adding auxiliary verification links ("Jump Links"), which 'jumps' an exponential distance of nodes.

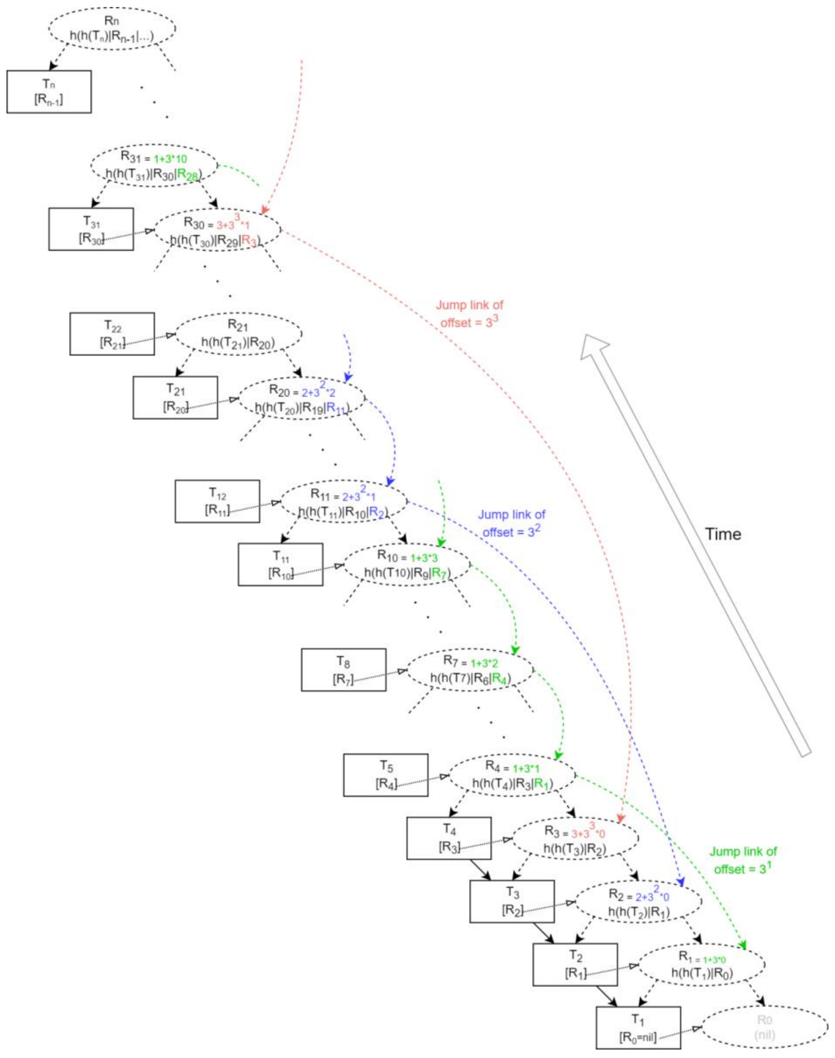
Jump link of offset = 3^3
Jump link of offset = 3^2
Jump link of offset = 3^1
Time

Figure 5 Hierarchical SMT of base=3

Jump link

An SMT is a binary Merkle Tree. Each label of Merkle Tree node is a hash value delivered from the label of its children. In SMT, the first child of a node has the hash value of new data block T_n, and the second child has the label of previous root node R_{n-1}.

The node label (hash value) of an SMT node *n*: $R_n = h(\ h(T_n)\ |\ R_{n-1}\ )$

An H-SMT node has one more child. The 3rd child is a link to a Merkle Tree node in a past point. We denote this third child as a "jump link."

The node label of an H-SMT node *n*: $R_n = h(h(T_n) \mid R_{n-1} \mid R_{(n-base^{offset})})$

Jump Distance

A jump link refers to a Merkle Root hash value of a past node. It is technically possible to use any node in the past. In Locus Chain, the relative position to the jump linked node (or the "jump distance") is determined based on the predefined "*jump base*" value and the height of the graph.

In particular, a jump distance is a multiple of the jump base distance. The minimum jump distance will be *base* and the maximum distance will be $base^{base}$.

Adding Jump Links

Let us have an example of building an H-SMT by adding jump links of base=3 on an existing SMT.

The jump distance is calculated based on the node *offset* to the *base*. The *offset* is the reminder of node height *n* divided by the *base*.

$offset = n \bmod base$

Then, the jump distance is the power of *base* to *offset*. If offset=0, then the distance is power of *base* to *base*.

$$jump\_distance = \begin{cases} base^{offset} & (\text{if } offset>0) \\ base^{base} & (\text{if } offset=0) \end{cases}$$

Figure 3 is an example of adding jump links to nodes of offset=1. The label of node 4 is delivered from three children, T4, R3, and R1.

$R_4 = h(h(T_4) \mid R_3 \mid R_1)$

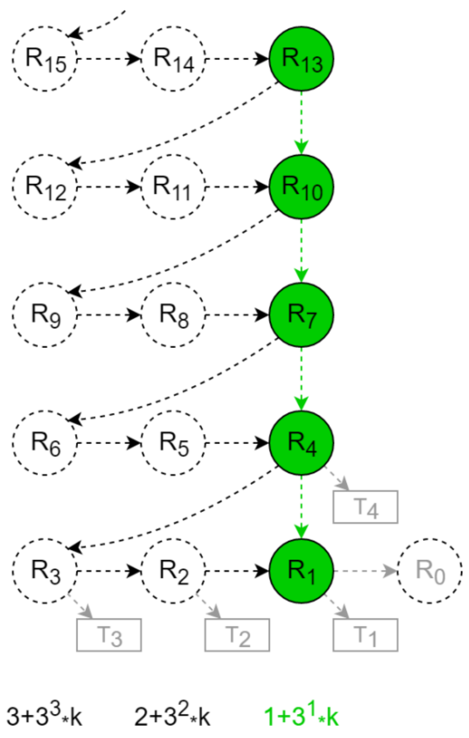

Figure 6 Jump link of nodes of offset 1 (base=3)

Next, let us place the jump links for nodes of *offset*=2. The jump distance of nodes of *offset*=2 is 3^2=9. In practice, it could be efficient to omit jump links if the link distance became long. In figure 4, the jump links for offset=2 are only placed on the nodes of height (*offset+baseoffset*k*) to form a directly linked list.

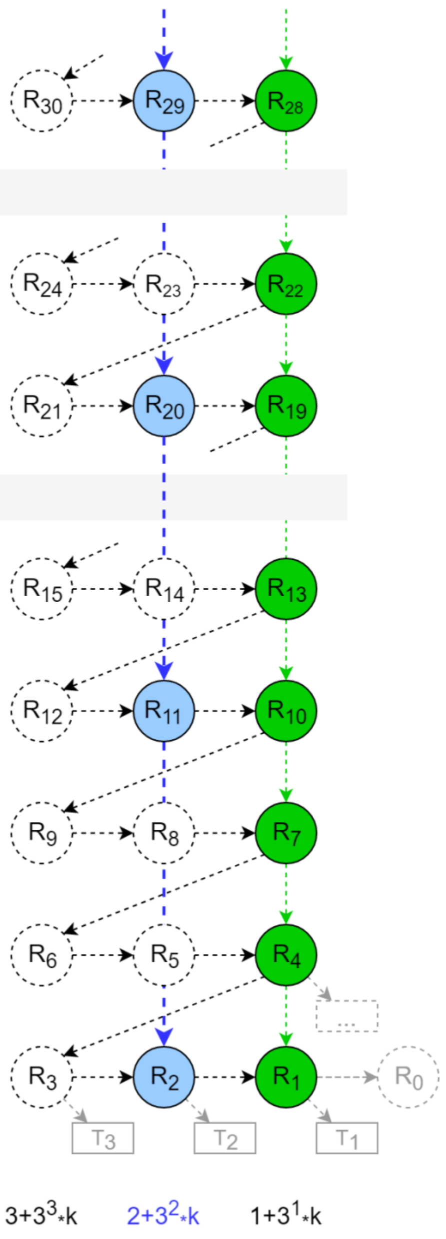

Figure 7 Jump Links of offset=2

The procedure is completed by repeat the steps until the *offset* reaches the *base*. Figure 5 is the final result of jump links of base=3.

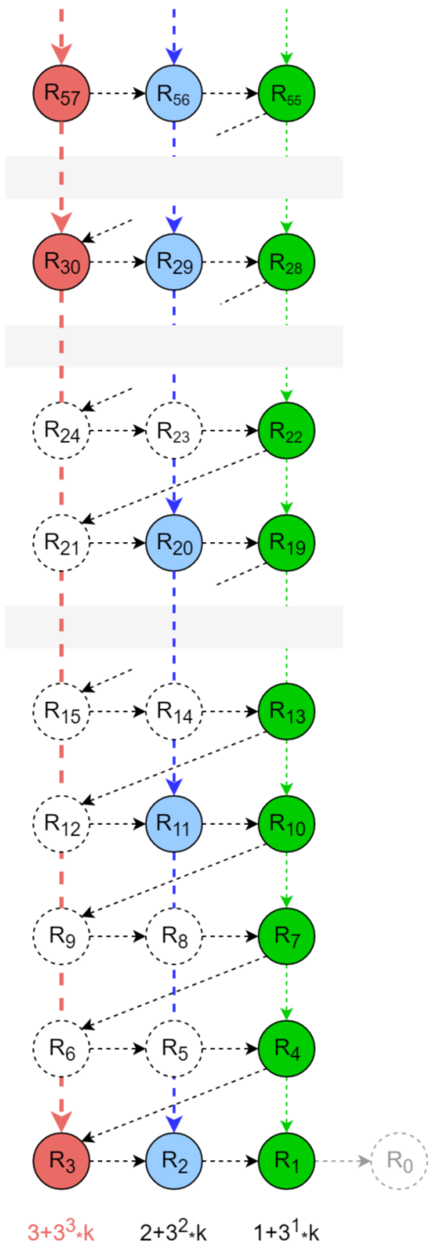

Figure 8 A example of Hierarchical SMT of base=3

Adding a new data block

The same algorithm applied to adding a new block T_n to an existing SMT. The offset and the jump distance are calculated using the height, then the hash label for the new root H-SMT node is calculated based on $h(T_n)$, R_{n-1}, and $R_{n\text{-jump\_distance}}$.

$R_n = h(\ h(T_n)\ |\ R_{n-1}\ |\ R_{n\text{-jump\_distance}}\ )$

Verifying a block

Assume that a verifier has a hash value R_y of a node in an H-SMT list, and the verifier wants to verify that a preceding block T_x, or hash of the block $h(T_x)$, is also contained in the H-SMT. This is a problem of finding a path from R_y to R_x, which is the direct parent of T_x. The verifier could follow the procedure below to find the path from R_y to R_x.

1) Beginning with Ry, examine nearby nodes { R_{y-1}, R_{y-2}, …, $R_{y\text{-base}}$ } to find the longest jump link that points to a node R_i (i>=x).

2) Record the *i* and repeat step 1 with the R_i until the *i* reaches x.

3) Calculate the chain of hash values for all found nodes in step 1 and 2. If the final value matches R_y, then it is true that T_x is a precedent of T_y.

Figure 6 is an example of finding a precedent block T_8 from node R_{59}. The immediate previous links are used for $R_{59} \rightarrow R_{58} \rightarrow R_{57}$. From R_{57} to R_{30}, a jump link of distance 27 is traversed. Jump links of distance 9 are used for R_{29} to R_{11}. Finally, R_{11} to R_8 is traversed.

The number of links traversed in this example is 9. In SMT, 59-8 = 51 traverses are required.

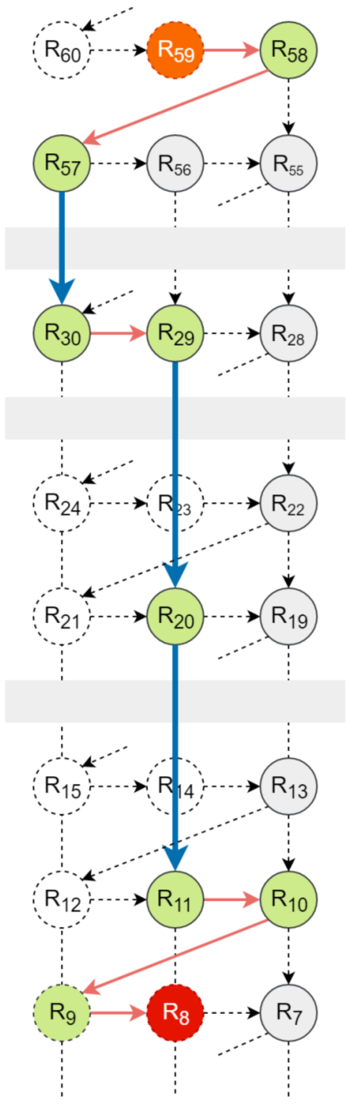

Figure 9 Traversing R59 to R8

Real-world Implementation

W uses the base of 3 in previous examples; however, a larger base value is suitable for real-world usages. Locus Chain uses a base value of 10 for normal circumstances, and the jump distance of base=10 ranges 10 ~ 10^{10}. For example, an optimal number of link traverses for two nodes of 1999 distance will be 28, delivered from (1*1000 + 9*100 + 9*10 + 9).

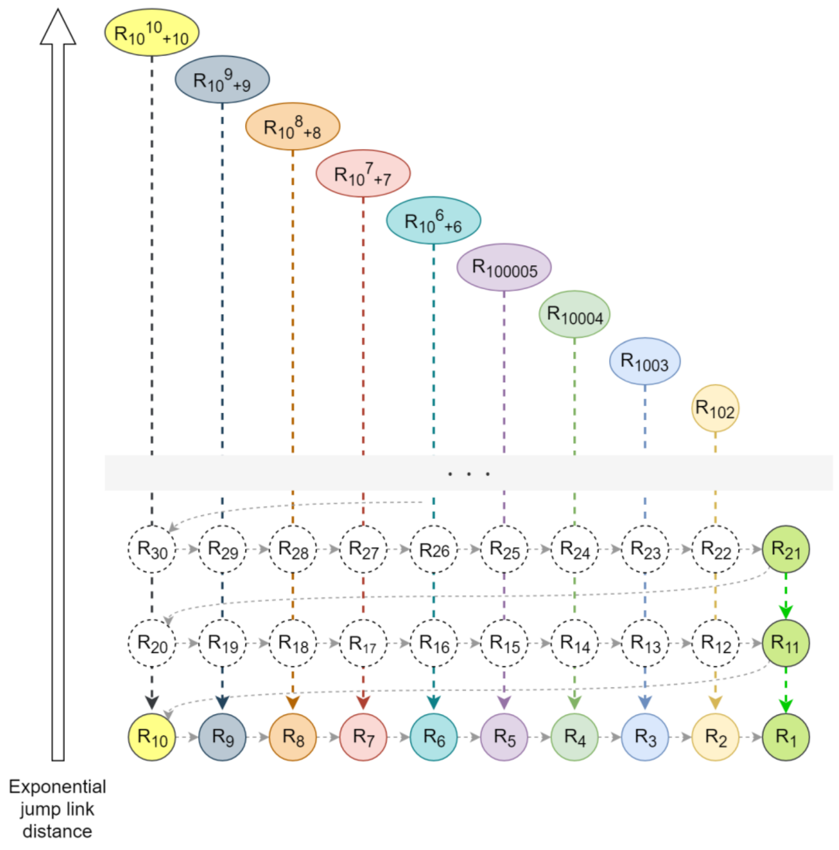
$R_{10^{10}+10}$
$R_{10^{9}+9}$
$R_{10^{8}+8}$
$R_{10^{7}+7}$
$R_{10^{6}+6}$
R_{100005}
R_{10004}
R_{1003}
R_{102}
. . .
R_{30} R_{29} R_{28} R_{27} R_{26} R_{25} R_{24} R_{23} R_{22} R_{21}
R_{20} R_{19} R_{18} R_{17} R_{16} R_{15} R_{14} R_{13} R_{12} R_{11}
R_{10} R_9 R_8 R_7 R_6 R_5 R_4 R_3 R_2 R_1
Exponential jump link distance

Figure 10 Locus chain's H-SMT with base=10

Detecting Forged Links

A malicious actor may forge hash values of H-SMT nodes by using illegitimate jump links. In H-SMT, such forged node values could be identified by cross-verifying multiple paths to the node.

An H-SMT is a direct acyclic graph, and if the distance between two nodes is longer than the base value, then there always exist two or more paths between the two nodes. Each path between the nodes is a Merkle Tree, and the calculated hash values of each Merkle Tree must match. If the forged node references an illegitimate hash value, then the hash value between paths would not match.

By this property, a node could be verified by selecting two different paths to a random distant precedent and comparing the path's hash values.

Summary

Verifying long chains of data have been computing-intensive overheads of blockchains. Some blockchain systems require a preparation time of multiple hours to verify the integrity of ledger data. Locus Chain cleared the problem with Hierarchical Skewed Merkle Tree, an innovative data structure that reduces data chain verification overhead to a logarithmic scale of chain lengths.

7. Networking

Locus Chain is a decentralized and distributed cryptocurrency system. Accordingly, Locus Chain assumes that general computers with ordinary performance communicate over the Internet to participate in the Locus Chain Network. Computers on the Internet that participate in Locus Chain voluntarily act as communication nodes to form a virtual network.

Although networking itself could be considered as an independent low-level layer separated from Locus Chain's core algorithm, it is an essential component for achieving performance goals.

(1) P2P Network

Similar to many other cryptocurrencies that incorporate the P2P system, the Locus Chain network uses the distributed network as a basis of communication, without any fixed central server. The communication between nodes currently uses the UDP.

(2) Node ID and Node distance

Each node in the Locus Chain Network has a host account (or a representative account). Each node generates its node ID by using the host account's address.

There is a concept of distance between two nodes. A node distance is a value calculated using bit-images of ID of two nodes, which tends to have short distance for adjacent IDs. Node distance is a numerical value unrelated to the real-world state of the physical network. Node distances are used to determine adjacency between nodes.

(3) Gossip Protocol

Locus Chain's distributed P2P communication fundamentally does not use direct communication among nodes. Instead, Locus Chain uses the protocol known as Gossip protocol.

There is no fixed route of communication in the Gossip protocol. Each node unconditionally transmits the received data to all connected nodes. The data is expected to be spread to all connected nodes eventually at some point in time, if nodes sufficiently repeat this transmission.

Each node in Locus Chain operates in the same way. When one node receives a data packet that has not been received in the past, the node transmits it to all connected, adjacent nodes. The data is ignored if the packet has been received in the past.

Nodes operate in the same way for new data. When a node produces a new data packet, the node transmits the produced data to all adjacent nodes.

The adjacent nodes that receive the data packet respond to the sender node with a confirmation message. Data transmissions are considered successful when the sender node receives confirmation messages from a sufficient number of adjacent nodes.

8. Dynamic Sharding

(1) Introduction to Locus Chain Sharding

Practical Bottleneck of Blockchain Scaling

Locus Chain's design goal is to provide practical Blockchain infrastructure suitable for real-world usage. We mean the goal of "real-world usage" as the capacity of transactions comparable to actual commercial transactions, like Credit Card transactions. Locus Chain's assumed capacity is at least 4,000 Transactions per Second, almost the capacity of the VISA. Also, we mean the goal of "practical" as the hardware and software requirements for the users. Locus Chain's assumed hardware base for Smart-Contract Enabled Nodes is today's regular consumer-grade PC. The requirement for transaction-only nodes is even low, as IoT-grade microdevices.

In short, Locus Chain aims to be a Blockchain system with 4k-TPS capability on PC, IoT devices, and Mobile Phones.

From the viewpoint of computing resources, the required resources for the Locus Chain's requirements are quite clear. When the capacity grows, the required resource also grows. Transactions require bandwidths for communication, CPU times for processing, and storage for histories, or the "Ledger".

Let us look into the bandwidth first. Assume that the size of an average transaction is 500 bytes (or 0.5Kbytes), then the raw data for 4,000 transactions per second is 4,000 * 500 bytes, or 2 Million bytes per second. This naive amount of data is about 20% of a typical 100Mbps home network. Actual blockchain operations may require 5~10x data, about 20MB/sec, for retransmissions and blockchain book-keepings. Indeed 20MB/sec bandwidths are available on today's high-end home networks but generally not feasible for average users.

The CPU requirement is fortunately ignorable. Locus Chain adopts PoS-based BFT consensus and does not require high CPU calculation for crypto puzzles. The minimally configured Locus Chain could run on ARM-based mini devices, which cost less than $10.

The Storage requirements are the toughest problem. Four thousand transactions per second require 2M bytes for every second, which add up to 172Gbytes of data per day. Today's average PC has about 500GBytes of SSD storage, and 172Gbytes per day fills up the entire storage in less than three days.

In summary, the bandwidth and storage requirement is the bottleneck to a practical blockchain. These understandings are commonly shared in the leading-edge blockchain development teams and tackled with basically the same approach; "Sharding".

General Sharding Approach

Sharding is a divide-and-conquer approach to the data volume problem. Sharding means dividing the system into “shards,” which are multiple independent sub-parts in blockchain systems. Then the data are divided and processed by each shard.

There are different sharding techniques in the viewpoint of resources divided.

Network sharding is about diving communication network. The communication network is divided into several sub-networks, “network-shards,” and transactions generated in a shard are typically processed within the specific shard. The network sharding typically reduces the bandwidth consumption of each node by reducing the number of communications.

Ledger sharding is about dividing the stored data. The ledger, which effectively is blockchain history, is divided into multiple sub-ledgers and stored in different sub-networks. The ledger sharding typically reduces the storage spaces of each node by reducing the amount of stored ledger data.

Consensus protocols could also be shared and may improve the transaction throughput of the whole blockchain by processing transactions in parallel.

These techniques could be applied together to enhance overall performance. For example, ledger sharding may incorporate network sharding to keep the ledger data in a certain shard. The consensus protocol may run in certain ledger shard to minimize communication regarding block generation protocol.

Pitfalls of Sharding

While the sharding approach is the proper way to solve the data volume problem, the sharding may introduce new issues to a blockchain system.

One of the typical issues related to ledger sharding is data synchronization. When a ledger is sharded into independent sub-ledgers, then the shards recognize no direct knowledge regarding other shards. In other words, a user in a shard cannot directly determine a piece of information regarding other shards is true or not. This issue is typically covered by introducing a super-blockchain, a manager blockchain that keeps track of sub-ledgers. Yet, there could be circumstances that the super-blockchain alone should be an insufficient solution for synchronizing inter-shard transactions that modify data across multiple shards.

A well-known issue regarding network sharding is a security problem. When the whole network is divided into N shards, then the number of nodes in each sub-shard is effectively reduced by N. The reduced number of nodes also reduces the required number of nodes to attack a blockchain network. So sharding reduces the overall stability of the network. Essentially there is a tradeoff between transaction throughput and network security, depending on the number of shards.

The issue becomes worse if the shards have a different number of nodes. An attacker may concentrate on a shard with the smallest nodes to take control of the shard. A simple solution to this situation is to increase the total number of nodes; however, it is not a technical, guaranteed solution.

Lochs Chain’s Approach

From the beginning, the design philosophy of Locus Chain is based on observations regarding blockchain scaling. There has been a clear technical limit of bandwidth and storage requirements for high-throughput blockchain, which leads to the sharding design of the Locus Chain. There has been a clear requirement regarding the security of the Locus Chain, which leads to a definite low-bar hardware requirement to make people involved.

The design of the Locus Chain starts from the lowest level, the ledger structure. Locus Chain's ledger is a composite of Account-Wise-Transaction-Chains (AWTC). AWTC is a Directed-Acyclic-Graph (DAG) based data structure designed for ledger sharding and resizing.

Locus Chain's sharding is Dynamic Sharding, and the configuration of shards changes on the fly. The number of shards changes dynamically based on the transaction throughput and the number of nodes. The numbers of nodes between shards are balanced dynamically to keep the security balance between shards. Nodes are moved between shards dynamically to introduce stability to the entire network.

Locus Chain incorporates Verifiable Pruning as a solution to the storage space problem. Nodes may prune data with no direct interest from the ledger to minimize storage spaces, and the ledger’s integrity is still verifiable. The pruned data could be retrieved by a data-query protocol if required.

With the nature of sharding and pruning, unknown ledger data exists from a node's perspective. Locus Chain's data and network layer provides built-in support for retrieving and verifying unknown, sharded, and pruned data, like data-query protocols and inter-shard communication channels. The communication scheme also works as a basis for inter-shard smart contracts.

(2) A Recap of Locus Chain Ledger Structure: Account-Wise-Transaction-Chain

The AWTC, Account-Wise-Transaction-Chain, is a central data structure for Locus Chain's high-capacity, distributed transaction processing.

AWTC is a Directed-Acyclic-Graph (DAG) based data structure composed of multiple transaction chains, each for an account. Each account has its dedicated chain. A new transaction issued by an account is primarily added to the account's chain.

Typical linear ledgers have only one spot for a new block to be connected to the previous block, which may become a bottleneck for high-speed data processing. In contrast, AWTC's DAG-based structure has multiple parallel data insertion points at any time, and there is virtually no limit of transactions added at a time. Also, collisions do not occur because a new transaction is appended to single fixed point on the AWTC graph and settled immediately if the issuer account is not malicious. This characteristic fundamentally resolves the transaction processing delay issue of previous blockchains.

Typical linear ledgers have only one spot for a new block to be connected to the previous block, which may become a bottleneck for high-speed data processing. In contrast, AWTC's DAG-based structure has multiple parallel data insertion points at any time, and there is virtually no limit of transactions added at a time. Also, collisions do not occur because a new transaction is appended to one fixed point on the AWTC graph and settled immediately if the issuer account is not malicious. This characteristic fundamentally resolves the transaction processing delay issue of previous blockchains.

A Transaction (Tx) is a unit of data added to the ledger. When an account wants to add information to Locus Chain's ledger, the account issues a transaction containing the information. Each transaction is issued solely by a single account. The issuer account signs the transaction with its secret key.

There are two ways to identify a transaction; hash value and index number. A Transaction Hash Value is a calculated result of a cryptographic hash value of a transaction's entire data. A hash value is suitable for pointing to an arbitrary transaction in the ledger.

A Transaction Index Number is a unique, sequentially increasing id number for transactions of a specific account. The first transaction of the account's AWTC is a Tx with index number 0(zero). Each consequent transaction has an index number increased by one from the previous transaction. By this, a transaction could be identified by the account address and index number.

The index number clearly specifies the order of transactions of an account. A set of transactions of an account, with its order preserved, is called a Transaction Chain. An account cannot delete or modify the previously issued transactions. The account can only 'extend' the chain by issue a new transaction.

Each account has only one transaction chain, and Locus Chain's ledger is equivalent to the sum of all account's transaction chains.

An account cannot manipulate another account's chain. However, the account may issue a transaction regarding requests to other accounts to its Transaction Chain. Such transaction acts as a message from an issuer account to other destination accounts. When a message transaction is propagated and reaches the destination accounts, then the destination accounts may issue transactions referencing the message transaction.

Let us see an example. Assume that account A wants to transfer some coins to account B. First, account A generates a transaction like {A: -10 from self, +10 to B}, adds to A's chain, then broadcasts it over the network. At this point, A's coin is reduced by 10, but B's coin is not changed. When B observes the A's transaction, then B completes the transfer by issuing a transaction like {B: +10 received from A} referencing the A's transaction. Now the B's coin is increased by 10, and the transfer completes.

The first transaction (0th transaction) of each account means the new creation of the account. 0th transaction always occurs as a coin-receiving transaction from another account. All accounts except for the genesis system accounts are derived from other accounts.

In summary, AWTC is a fundamental basis for Locus Chain's high-capacity transaction processing.

(3) Locus Chain’s Dynamic Sharding

Locus Chain changes the configuration of shards on the fly to optimize the performance of the whole system. This reconfiguration is called Dynamic Sharding.

The shards are split and rebalanced if required. Shard splitting is the action of dividing a shard into two shards. Shard rebalancing is an action of moving nodes between shards.

Basic Structure of Shards

Locus Chain implements network sharding and ledger sharding in harmony. Each node is assigned to a particular shard at a time. A node in Locus Chain manages a set of accounts and its transaction chains. Then, the shard's ledger (shard-ledger) is the sum of transaction chains of all nodes in the shard. Each shard has exclusive control of a shard-ledger. The number of shard ledgers is equal to the number of network shards. Shards are identified by an ID number.

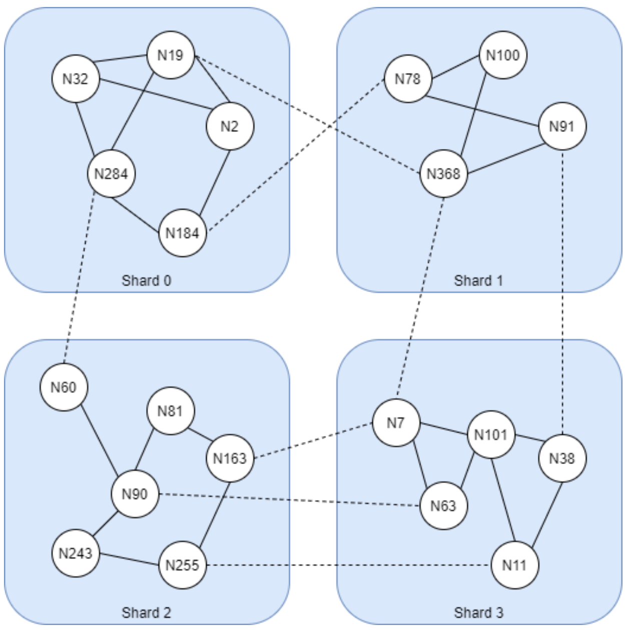

Figure 11 Conceptual Dialogram of Locus Chain Shards

Shard Reconfiguration

Locus Chain determines the designated number of shards using a criterion for optimizing the whole system's performance. If the designated number of shards is different from the current number, then the shards are split.

As the criterion of the shard count, Locus Chain evaluates the network loads and computational loads of some periods.

Locus Chain builds on a Peer-to-Peer network substructure. The network load could be estimated based on the number of nodes and the density of connections. The shards could be reconfigured by shard splitting or shard rebalancing if the network loads of shards are out of a specific range.

Shards are rebalanced when the number of nodes or the number of accounts between nodes becomes biased. Reconfiguration of shards is an essential operation for improving the stability of the whole system.

The reconfiguration is executed on an epoch basis. Currently, an epoch is changed daily, and the shards splitting or shard rebalancing is also decided and executed daily on epoch change.

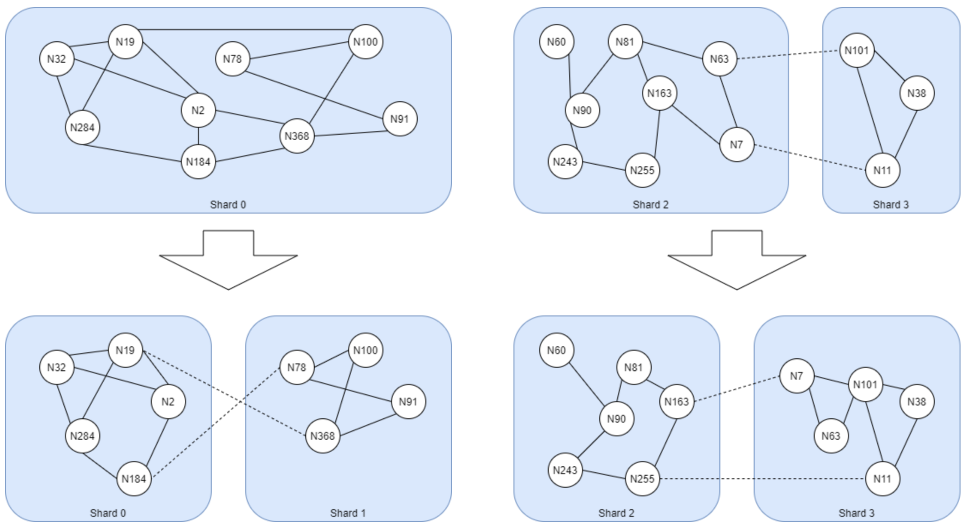

Figure 12 Shard Dividing and Rebalancing

Inter-shard communication network

Typically a transaction is propagated only to the nodes in the same shard. However, when a message transaction has a destination address that belongs to another shard, the message must be passed to the destination shard. For this, sharding introduces another layer of abstraction of a communication network. The traditional P2P network in a shard is named "intra-shard network layer," and the new network is named the "inter-shard network layer," which exchanges information between shards.

Each network node peer participates in both layers simultaneously. A specific portion of P2P connections of a node is used for the intra-shard layer to communicate with nodes of the same shard, and the rest is used for an inter-shard layer to communicate with nodes of other shards.

The inter-shard network uses the same p2p communication strategy as the intra-shard layer. The difference is the list of nodes. The inter-shard layer uses a list of nodes in other working shards. However, there is a restriction about the shards that could be connected. A node can only connect to the nodes in working shards in which the shard ID differs only one bit from the node's working shard ID. For example, let us assume four working shards in the world with IDs ranging from 0 to 3. Each shard (actually a node in the shard) could connect to another shard where the two shard IDs are different in only one bit. For the case of shard #0 (00 in binary), flipping any one-bit of shard ID results in #1(01 in binary) and #2(10). Thus the shard could make connections with shard #1 and shard #2. Similarly, shard #1(01) could communicate with #0(00) and #3(11). Respectively, shard #2(10) communicates with #0(00) and #3(11), and shard #3 communicates with #1 and #2.

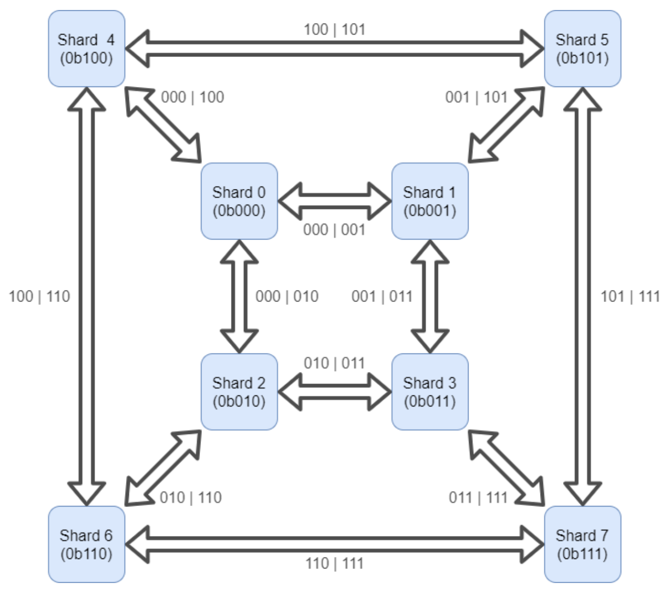

Figure 13 Inter-shard communication network of 8 shards (3 bits)

For shards that are not directly connected, intermediate shard relays the communication. For example, when shard #0 communicates with shard #3, shard #0 sends information to shard #1 (or shard #2), then the nodes in shard #1 (or shard #2) relays the information to shard #3.

In this way, inter-shard communication forms a kind of fixed-relay P2P communication network between shards. A shard has to keep a list of nodes in other shards to initiate inter-shard communication. Because the connections between shards are virtually fixed, the node list's size could be reduced to the number of nodes in adjacent shards.

World Round Status

Each shard executes a consensus algorithm independently. That means each shard has its own Round State Chain that keeps track of the shard's ledger.

Since the information in each shard is independent and invisible to other shards, we need a mechanism to integrate individual shard states into one global system state to keep the integrity of the whole world. The World Round State Chain (WRS Chain) mechanism is introduced for this.

WRS Chain is a unique chain consists of World Round States.

Each World Round State contains representative states of each shard and meta-information regarding the world. The meta-information includes values such as 'round number,' 'current number of shards,' 'number of shards at the next round,' and 'list of nodes that enter or leave each shard.' Shard states are stored as Merkle root hashes of each shard's round state. (Please be note that actual stored Merkle hash values are of the previous round.)

A node could use Merkle root hashes in WRS to test the integrity of information from other shards. For example, a node could prove the integrity of a transaction to other shards by supplying intermediate hash values related to a Merkle root hash that is publicly acquirable from the WRS Chain, along with the transaction.

A World Round State (WRS) results from an agreement for each round, like a shard round state. An existing shard consensus committee also executes a WRS consensus. For each round, a random shard is selected for WRS consensus; then, the selected committee executes a WRS consensus, as well as the shard's round state consensus.

9. Smart Contract on Locus Chain

Locus Chain implements several technical breakthroughs that enable high-capacity transaction processing on decent machines of limited network bandwidth, data storage, and low computing powers. Our Dynamic Sharding Technology reduces required network bandwidth and computing energy consumption, even suitable for lightweight IoT devices. Our Verifiable Pruning Technology saves data storage requirements and initial startup time while ensuring ledger integrity. These breakthroughs are backed by a modular DAG data structure named Account-Wise-Transaction-Chain.

However, Locus Chain's sophisticated data management technologies introduced new problems for the Smart Contract execution environment. The ledgers and computing nodes, partitioned by Dynamic Sharding technology, cannot directly reference and verify transactions between shards. Verifiable Pruning makes low-capacity machines like IoT devices eligible to Locus Chain, but executing Smart Contract codes may be a difficult task for such machines.

We are actively developing an improved Locus Chain Smart Contract system that addresses these new difficulties while maintaining compatibilities with today's popular Smart Contract systems.

(1) Locus Chain Smart Contract Account

Technically, Locus Chain's Smart Contracts are accounts with a program code and its runtime state information. Like a regular user account, a Smart Contract Account (SCA) has an AWTC transaction chain to issue transactions. Unlike a regular account, the transactions are issued by automated smart contract evaluation facilities, or Smart Contract Execution Committees.

Any account can start a Smart Contract by creating a Smart Contract Account. An account initiates an SCA by issuing a Smart Contract creating transaction with initializing program code and initial funds. Smart Contract Execution Committee evaluates the Smart Contract Creating Transaction and then initiates the first transaction (TX0) of the new SCA, containing the main program code and proposed funds.

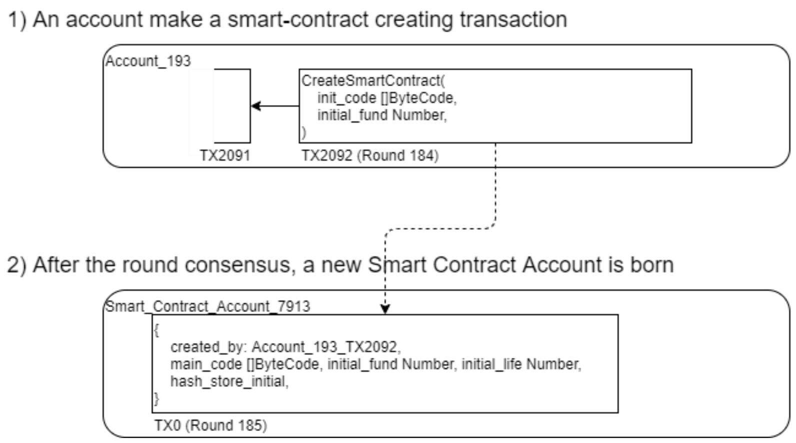
1) An account make a smart-contract creating transaction
Account_193
CreateSmartContract(
init_code []ByteCode,
initial_fund Number,
)
TX2091
TX2092 (Round 184)
2) After the round consensus, a new Smart Contract Account is born
Smart_Contract_Account_7913
{
created_by: Account_193_TX2092,
main_code []ByteCode, initial_fund Number, initial_life Number,
hash_store_initial,
}
TX0 (Round 185)

Figure 14 Initializing a Smart Contract

The main program code of an SCA is executed (or triggered) by transactions from other accounts. For example, if the SCA's main program has a 'callback' function for coin-sending transactions, then the callback function will be executed every time another account sends coins to the SCA. The SCA's main program may append new transactions to its AWTC as a result of the computation.

Smart Contract Accounts are not permanent. Every active SCA has a positive “life” value, and the life value is reduced according to its activity. The SCA becomes inactive (or terminated) when the life value reaches zero.

An SCA’s life is a value exchangeable for coins, and the reduced life is a cost of management for the Locus Chain system. The reduced life is collected by accounts participating in the consensus as incentives for the consensus works. Other accounts may extend an SCA’s life by supplying funds.

The transactions of live SCAs are not pruned. However, inactive (or terminated) SCAs may be pruned immediately.

An SCA’s main program may terminate itself at any time.

(2) Contract Evaluation Node / Computing Node

Locus Chain runs well in lightweight machines like IoT devices. However, Smart Contract programs may require high CPU loads not affordable for low-powered machines. Because of this limitation, Locus Chain does not mandate nodes to participate in the Smart Contract Consensus process. The machines with excessive CPU powers are encouraged to participate in the Smart Contract Consensus and collect additional incentives.

The nodes that participate in the Smart Contract are called Contract Evaluation Nodes, or just Evaluation Nodes. Evaluation Nodes also handle typical roles like ledger management and consensus committee.

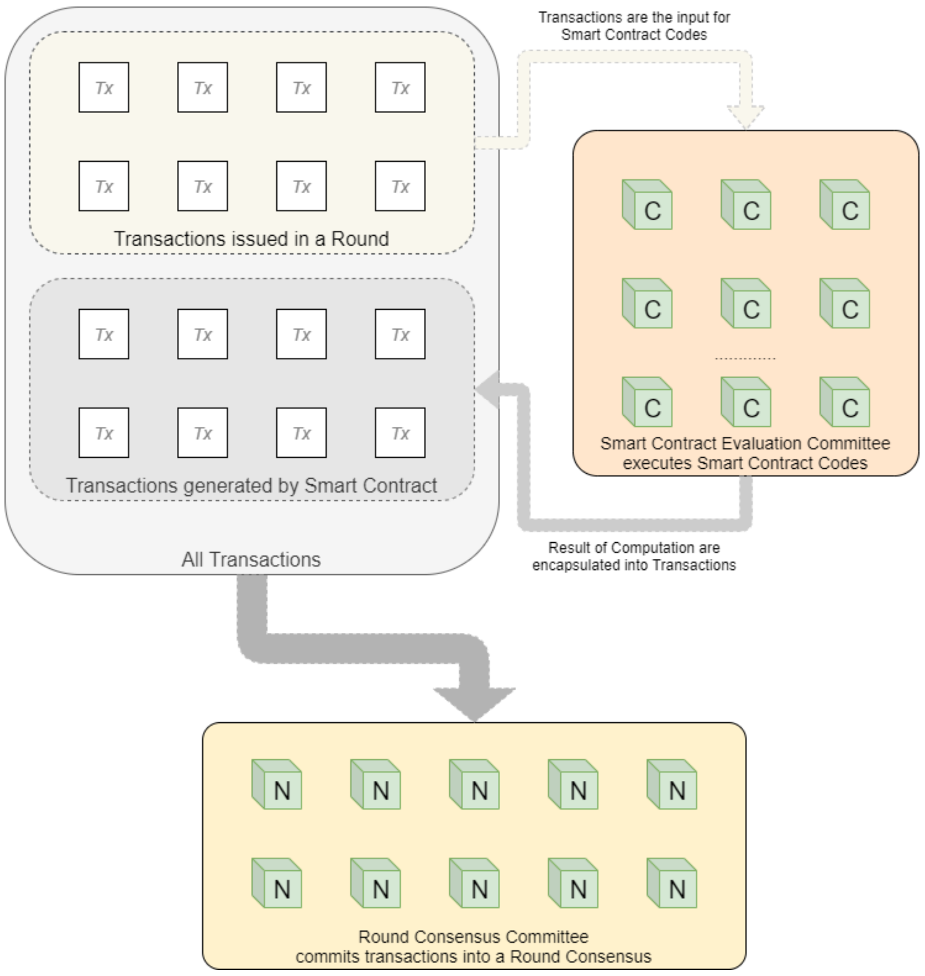
Transactions are the input for Smart Contract Codes
Tx
Transactions issued in a Round
Transactions generated by Smart Contract
All Transactions
C
Smart Contract Evaluation Committee executes Smart Contract Codes
Result of Computation are encapsulated into Transactions
N
Round Consensus Committee commits transactions into a Round Consensus

Figure 15 Smart Contract Evaluation Committee & Consensus Committee

Evaluation Nodes form an independent committee named "Evaluation Committee." The Evaluation Committee accepts transactions regarding Smart Contracts and executes Smart Contract programs accordingly. The computed results are packed into SCA's transactions and passed to Consensus Committee to form a round's consensus.

As the AWTCs are independent and parallel data structures, different SCAs could run in parallel. The order of computation between different SCAs usually does not matter. However, the order of transactions regarding the same AWTC affects the computing result. The transactions in the same chain must be sorted in fair, proper order for the reproducibility of results. Locus Chain determines the sort order is determined based on multiple aspects such as fuel price and transaction fees.

(3) Sharded ledger and Message-passing smart contract

The ledger of the Locus Chain is divided by Shards. Because of the sharding, nodes in a shard have no information regarding other shards; nodes cannot verify the legitimacy of message transactions from other shards by themselves. In Locus Chain, there are mechanisms for handling this limitation. One of the mechanisms is the inter-shard query protocol.

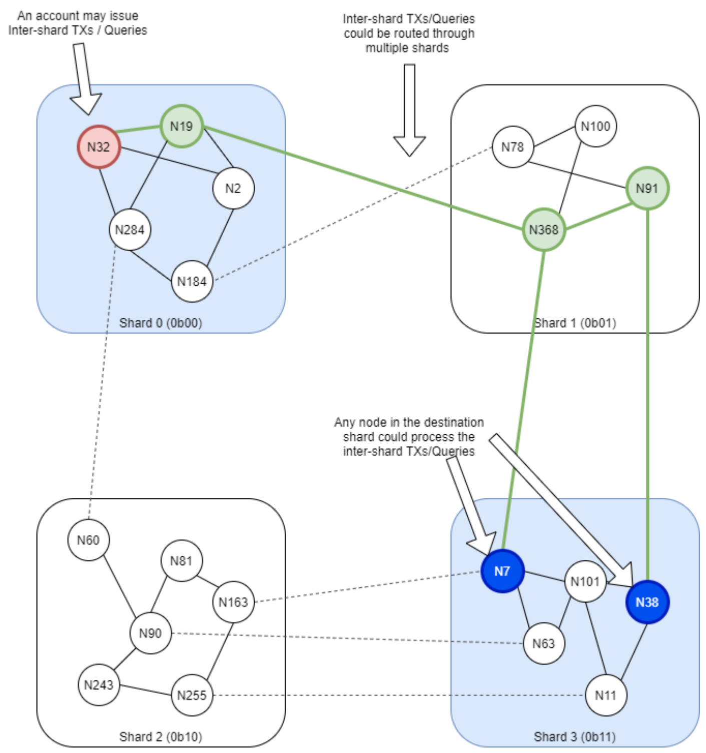
An account may issue Inter-shard TXs / Queries
Inter-shard TXs/Queries could be routed through multiple shards
Any node in the destination shard could process the inter-shard TXs/Queries
N32
N19
N2
N284
N184
Shard 0 (0b00)
N78
N100
N91
N368
Shard 1 (0b01)
N60
N81
N163
N90
N243
N255
Shard 2 (0b10)
N7
N101
N38
N63
N11
Shard 3 (0b11)

Figure 16 Inter-shard Transactions and Queries

As well, in Locus Chain's Sharded Smart Contract Model, the smart contract transactions use the same inter-shard query protocol to interact with accounts of other shards. The inter-shard query is transparent to an SCA; when a smart contract program issues a transaction, Locus Chain automatically invokes an inter-shard query if the recipient is not in the same shard. While the inter-shard query may take some extra time in the worst situations, the messages are guaranteed to be delivered regardless of sharding.

(4) Contract Structure Layer and Virtual-Machine Layer

Locus Chain's current Smart Contract model defines a set of interfaces as a blockchain system's layered component. The Contract Structure Layer is the base system of Locus Chain's Smart Contract model. The Contract Structure Layer defines aspects like Synthetic Accounts, the lifecycle of Smart Contract Accounts, transaction management for Synthetic Accounts, execution model of a Smart Contract.

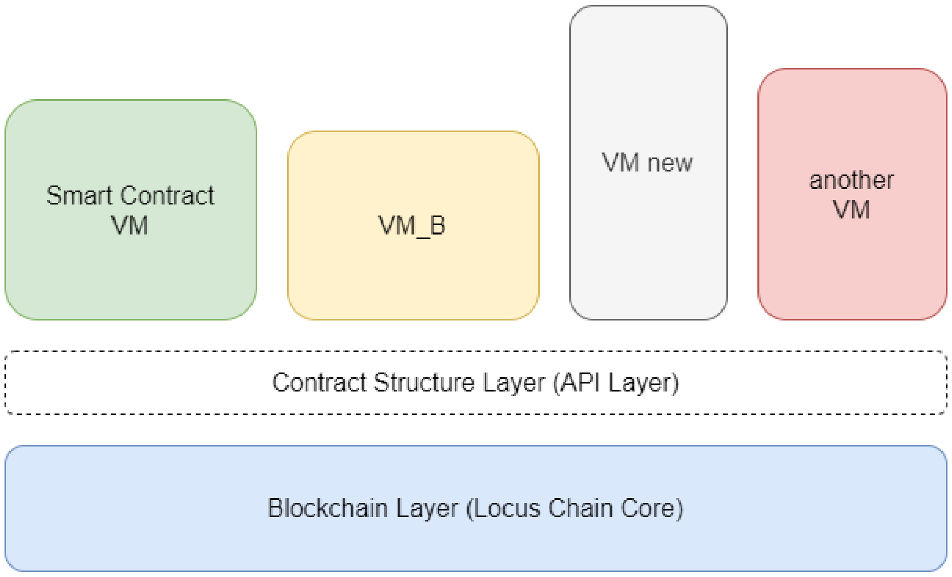
Smart Contract VM
VM_B
VM new
another VM
Contract Structure Layer (API Layer)
Blockchain Layer (Locus Chain Core)

Figure 17 Layered architecture of Locus Chain Smart Contract

Virtual Machines, like bytecode interpreters, are implemented on the Contract Structure Layer to run an actual Smart Contract program properly. There could be several different VMs implemented. The VMs are placed in Virtual Machine Layer. The VMs in the Virtual Machine Layer may interact with each other via the Contract Structure Layer. This separation of blockchain-level APIs and Virtual Machines enables flexible and extensible Smart Contracts configuration in Locus Chain. VMs could be highly customizable to use cases, and there could be compatible VMs with other blockchain systems.

Currently, Locus Chain implements a Solidity-compatible Ethereum-VM bytecode interpreter. Other VMs, like WebAssembly-based VM and functional-style VM, are also in our sight.

10. Cryptographic Keys

(1) Locus Chain Key System

Similar to many other blockchain platforms, each account of Locus Chain is related to a pair of public and secret key supplied by a user. Generally, users have to securely manage their secret key.

In Locus Chain, there are three kinds of cryptographic keys related to an account. There is a master key, a normal key, and a validation key.

A validation key is a kind of transient key for real-time use, such as signing messages for consensus processes. A validation key is automatically generated from the normal key. The validation key is supposed to be re-generated frequently.

A normal key is a key for general-purpose operations such as transaction signing. A normal key should be delivered by an algorithm balanced between cryptographic strength and ease of use. Currently, Locus Chain generates normal keys with an elliptic function cryptography algorithm.

A master key is a key for generating a normal key. A master key should be generated with a strong, computing-intensive algorithm usually not suitable for real-time operation. The algorithms for master keys are not finalized but working on both post-quantum cryptography systems and elliptic-function cryptography systems.

If a normal key becomes inappropriate for some reason, the user can discard the key and register a new key by issuing a key-exchange transaction with a new normal key generated by using the user's master key. The new normal key becomes valid for the next round of the consensus process which settles the key-exchange transaction.

Furthermore, Locus Chain is working on an approach for adding and changing algorithms for normal keys. We hope this approach could be a future-proof facility for the post-quantum computing era.

(2) Post-Quantum Cryptography

One of the biggest threats to blockchains is the emerging threat of quantum computers. There is a possibility that quantum computers may break many currently used signature algorithms. Yet, there are also researches regarding Post-Quantum Cryptographies (PQC) that are secure against quantum computers.

However, current Post-Quantum Cryptographies (PQC) are infeasible for personal computers and mobile devices because they require immense amount of computation and data compared to traditional algorithms. Also, there are no matured standards for PQCs yet, and it is hard to tell the precise property of PQC algorithms in actual use due to the lack of scientific and technological establishments.

To overcome this situation, Locus Chain incorporates a layered key system composed of two cryptographic keys, a master key and a normal key, of different strengths. A user uses the normal key, which depends on a current-generation cryptography system, for signing transactions. If the user's normal key is compromised, then PQC-applied master key is used to re-generate and change the broken normal key.

The algorithm for master keys can consume significant computing resources for PQC computation since master keys are only used in exceptional situations. Also, Locus Chain is working on a built-in mechanism to add additional signature algorithms. In the future, Locus Chain will be able to exchange the normal key algorithm itself with a Post-Quantum algorithm once PQC algorithms become widely available. Moreover, Locus Chain may use a hybrid of PQC and current-generation cryptographies for a while since the property of PQC is not yet completely understood. Even when vulnerabilities of PQC signature algorithms are found, Locus Chain can make use of current-generation algorithms to quickly, temporarily cover the vulnerabilities.

10. TOKENOMICS DESIGN

(1) Tokens

The token type before the main net will be an Ethereum-based ERC20 token with smart contract issued by Locus Chain Foundation based in Singapore. However, given that the tokens on the public Ethereum network cannot be used on the Locus Chain directly, we will be making use of hashed time-locked contracts to allow 1-to-1 atomic swaps of Locus tokens from the Ethereum main-net to the Locus Chain. Multiple examples of such cross-chain atomic swaps already exist, such as Republic Protocol, or COMIT. For simplicity, we will not attempt to integrate these protocols, and will only support 1-to-1 atomic swaps of Locus tokens from the public Ethereum main-net to Locus Chain, and vice versa. This ensures that there will be enough Locus tokens circulating on the public Ethereum main-net, for liquidity and tradability purposes.

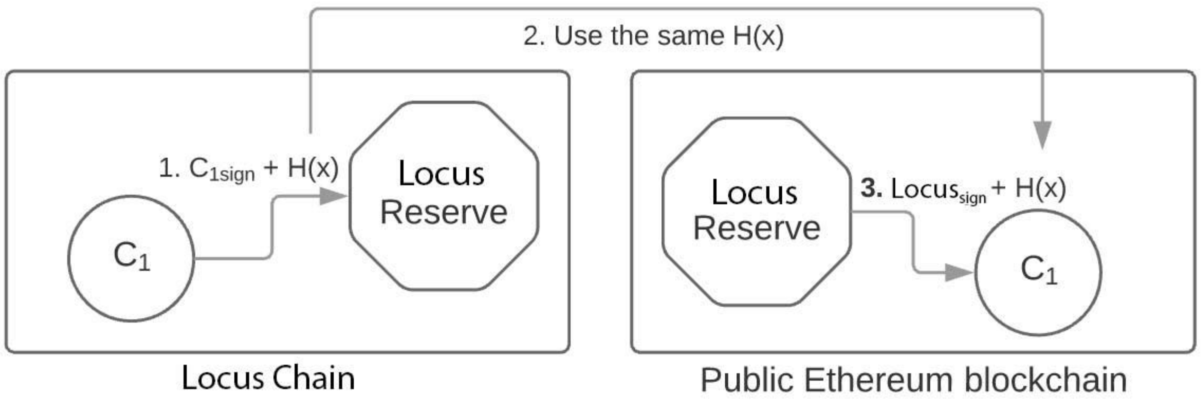

Figure 19 Atomic Swaps (Locus Chain <> Ethereum Main-net)

(2) Token Economy

The primary design of the token distribution and economic incentives were centralized around three concepts that act as foundational pillars that underpin the rest of the ecosystem value mechanisms:

- Value stability and security
- Incentive compatibility proportional to contributions to the ecosystem
- Decentralization of value aggregation and ownership

In order to fulfill these three criteria, Locus Chain’s consensus mechanism will use a form of Delegated Proof of Stake (DPOS).

The primary trait of the DPoS that we are using here is that the validator is not predetermined when the network is formed, but rather they’re autonomically determined by the participating nodes within the network. Within this system, anyone can become a “validator” by downloading the ledger and meeting the criteria that they have not delegated their stakes, and then maintaining connected stake long enough to be selected as a validator. This model is suited to nodes running background on desktop PCs normally connected to the internet.

When the node delegates its stake, the node is then classified as a “light user”. Light users do not mine coins but have the capabilities to transact without keeping the ledger, this model is thus suited for nodes running on hardware with lesser processing power like smart phones. This choice gives freedom to anyone to be a validator to mine Locus as long as they have accumulated a lot of stakes and have long internet access time, regardless of their hardware specifications and hash power.

The Locus Chain DPoS mitigates against asymmetric mining circumstances, as the barriers to mining Locus tokens are low, and anyone can do mining easily. The system ultimately only allows programmatic rules that can prevent from delegating errors or byzantine faults (malicious hacking attempts). Apart from these rules, any other artificial interventions aren’t allowed in the system, and this reinforces high levels of decentralization. Such a mechanism is better than traditional DPoS as it prices in the positive externalities of contributions to the ecosystem and network, which is currently undervalued.

Aside from the usual benefits of DPOS such as faster processing speeds and not needing to waste unnecessary energy on hashing, our incentive mechanism is also very effective for kickstarting stable and constant growth of the platform, which is a key factor that is a barrier for many new blockchain platform entrants. This is a gap that we have seen in other consensus mechanisms, which do not sufficiently factor in the high risks associated with early contributors, whereas our DPoS mechanism scales rewards based on the such associated contributions. By rewarding early contributors fairly for the risks associated with building on top of our new platform, we send a strong message that shows that we value platform creators and enablers that help to generate platform growth.

(3) Token Allocation

A total of 7 billion Locus Tokens has been pre-mined to be issued, with the entire supply of tokens to be allocated for private sales without any public participation. Should there be any remaining tokens that have yet to be sold before being listed, they would be sold as a form of block deal through over the counter transactions.

All of the foundation supplies are owned by the Locus Chain foundation and are specifically allocated only to causes with business benefits. The supply of tokens for advisors and partnership entities are mostly kept in lock-up for a period of 6 to 24 months.

Apart from the 7 billion Locus tokens, there are no more additional issuing except through mining after the launch of the main-net. Locus token would be fully atomic swapped to locus coins at the launch of the main-net.

Entity	**Token Quantity**	**Percentage**	**Remarks**
Foundation Reserves	1,200,000,000	17%	Locus chain foundation has and holds these which can be used for great strategic benefits and contributions.
Founders & Team	600,000,000	9%	Lock Up period of 2 years
Advisors & Partnerships	1,200,000,000	17%	Lock Up period of 6~24 months
Contributors	4,000,000,000	57%	
Total	7,000,000,000	100%	

Figure 20 Token Distribution Quantities

Additional mining of Locus tokens would only start after the launch of the main-net, with the maximum number of tokens to be mined capped at 5 billion Locus tokens. Based on estimates and tests run on the current mechanics, the 5 billion additional Locus tokens are to be completely mined up over an approximate period of 100 years. This would then fulfill the total token supply of 12 billion Locus Tokens.

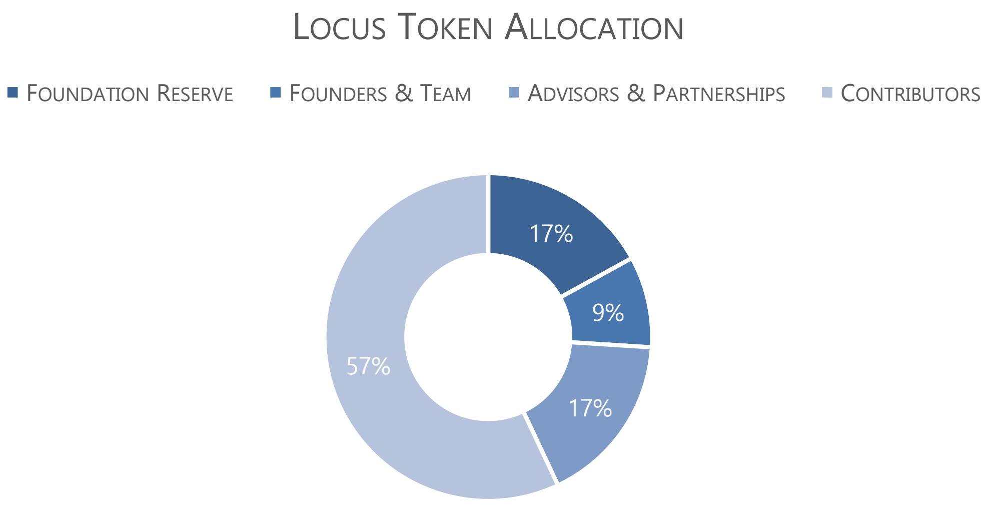

Figure 21 Locus Token Allocation

(4) Utilization of Funds Raised

The funds raised through the sale of Locus tokens would be used for the major operations of building up and developing the project. Technical development of the blockchain is core to the success of the project and utilizes 50% of the funds raised. Subsequently, 20% would be used to build strategic partnerships with entities that support the Locus chain ecosystem. 15% of funds raised would be used for marketing purposes to develop the public adoption of Locus chain. 10% of funds raised would be used to support the daily operational overheads of the project, and the last 5% of funds raised would be kept in reserve.

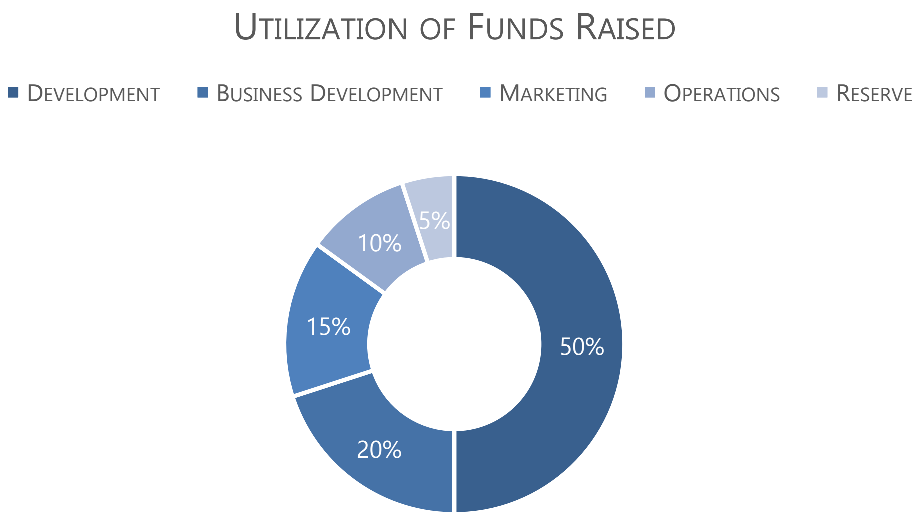

Figure 22 Utilization of Funds Raised Distribution

(5) Governance

"Locus Chain" has the following governance operation plans.

Based on the PoS policy, voting rights are granted according to the amount of coin shares held by each node. Currently, more voting rights are given to the operator than to the nodes that are delegated to secure network stability.

Voting rights are also delegated when the shares are delegated.

Various policy proposals, changes in policy, and details regarding voting will be made through the official wallet provided by the foundation. Details regarding initiatives, restrictions, voting, execution, and penalties will be announced later."

11. Summary

Locus Chain is an innovative blockchain platform to resolve performance and scalability issues without sacrificing the value of decentralization. Locus Chain is a high-speed, high-capacity blockchain system for diverging users on many kinds of hardware devices.

In this document, we described technical aspects that support the exceptional performance of the Locus Chain system.

Locus Chain adopted a nonlinear DAG structure to introduce parallelism and flexibility to the ledger. Locus Chain's Account Wise Transaction Chain (AWTC) structure is a parallel DAG data structure that describes relations between transactions. AWTC's multiple endpoints enable simultaneous high-speed referencing and processing of transactions, essential for Locus Chain's ultimate performance. AWTC's parallel nature makes ledger sharding a reality.

Locus Chain's consensus algorithm adopts the BFT consensus based on PoS (Proof-of-Stake). With the PoS-based BFT consensus, Locus Chain achieves fairness and correctness while reducing the inefficiencies of PoW and Nakamoto Consensus. Locus Chain's consensus elects consensus participating nodes fairly, considering various contributions of nodes to the Locus Chain network. The contributions include aspects like online time, storage space, computational power, and pure stake.

Locus Chain resolves several resource issues using Dynamic Sharding. Sharding reduces each node's data processing load by dividing ledgers. Sharding also reduces network communication load by localizing communications. Each shard independently executes the BFT consensus algorithm, and the total transaction throughput is multiplied by the number of shards. Locus Chain ensures fairness between shards by dynamically adjusting the number of nodes of shards in semi-realtime.

Verifiable Pruning is the key technology to the storage space problem. Verifiable Pruning drastically reduces each node's storage space requirement while keeping ledger integrity. Locus Chain could fully operate on devices with limited storage capacity like IoT devices.

Bibliography

Association, T. L. (n.d.). *An Introduction to Libra*. Retrieved from https://libra.org/en-us/whitepaper

Bentov, I., Gabizon, A., & Mizrahi, A. (2016). *Cryptocurrencies without proof of work.* Retrieved from https://arxiv.org/abs/1406.5694

Bernstein, D. J. (2015). Multi-user Schnorr Security, Revisited. *Cryptology ePrint Archive; Vol. 2015/996.* IACR.

Bitcoin: what a waste of resources. (2017, 11). Retrieved from New Scientist: https://www.newscientist.com/article/mg23631503-300-bitcoin-what-a-waste-of-resources/

Buterin, V. (n.d.). *Ethereum Sharding FAQ*. Retrieved from Ethereum Wiki: https://github.com/ethereum/wiki/wiki/Sharding-FAQ

Buterin, V., & Gavin, W. (2013). *Ethereum*. Retrieved from https://www.ethereum.org/

Castro, M., & Liskov, B. (1999). Practical byzantine fault tolerance. *Proceedings of the Third Symposium on Operating Systems Design and Implementation (OSDI'99)*, (pp. 173-186).

Chen, L., Jordan, S., Liu, Y.-K., Moody, D., Peralta, R., Perlner, R., et al. (2016). *Report on Post-Quantum Cryptography.* National Institute of Standars and Technology, Internal Report 8105.

Dang, H., Dinh, T. T., Chang, D. L.-C., Lin, Q., & Ooi, B. C. (2019). Towards Scaling Blockchain Systems via Sharding. *2019 International Conference on Management of Data (SIGMOD '19).* ACM.

Demers, A., Greene, D., Hauser, C., Irish, W., Larson, J., Shenker, S., et al. (1987). Epidemic Algorithms for Replicated Database Maintenance. *Proceedings of the Sixth Annual ACM Symposium on Principles of Distributed Computing (PODC '87)* (pp. 1-12). New York: ACM.

Directed Acyclic Graph. (n.d.). Retrieved from Wikipedia: https://en.wikipedia.org/wiki/Directed_acyclic_graph

Gilad, Y., Hemo, R., Micali, S., Vlachos, G., & Zeldovich, N. (2018). *Algorand: Scaling Byzantine Agreements for Cryptocurrencies.* Retrieved from https://algorandcom.cdn.prismic.io/algorandcom%2Fa26acb80-b80c-46ff-a1ab-a8121f74f3a3_p51-gilad.pdf

Lamport, L., Shostak, R., & Pease, M. (1982). The Byzantine generals problem. *ACM Transactions on Programming Languages and Systems, 4.3*, 382-401.

LeMahieu, C. (2017, 11). *Nano: A Feeless Distributed Cryptocurrency Network.* Retrieved from https://nano.org/en/whitepaper

Maymounkov, P., & Mazières, D. (2002). Kademlia: A Peer-to-Peer Information System Based on the XOR Metric. *International Workshop on Peer-to-Peer Systems (IPTPS 2002)*, (pp. 53-65).

Merkle, R. C. (1987). A digital signature based on a conventional encryption function. *A Conference on the Theory and Applications of Cryptographic Techniques on Advances in Cryptology (CRYPTO' 87)*, (pp. 369-378).

Merkle-Patrica Trie Specification. (n.d.). Retrieved from Ethereum Wiki: https://github.com/ethereum/wiki/wiki/Patricia-Tree#main-specification-merkle-patricia-trie

Micali, S., Vadhan, S., & Rabin, M. (1999). Verifiable Random Functions. *IEEE 40th Annual Symposium on Foundations of Computer Science (FOCS'99)*, 120-130.

Mills, D., Wang, K., Malone, B., Ravi, A., Marquardt, J., Chen, C., et al. (2016). *Distributed ledger technology in payments, clearing, and settlement.* Divisions of Research & Statistics and Monetary Affairs, Federal Reserve Board.

Nair, G. R., & Shoney, S. (2017). BlockChain Technology: Centralised Ledger to Distributed Ledger. *International Research Journal of Engineering and Technology, Vol.4, Issue 3*.

Nakamoto, S. (2008). *Bitcoin: A Peer-to-Peer Electronic Cash System*. Retrieved from https://bitcoin.org/bitcoin.pdf

Popov, S. (2018, 4). *The Tangle.* Retrieved from https://www.iota.org/research/academic-papers: https://assets.ctfassets.net/r1dr6vzfxhev/2t4uxvsIqk0EUau6g2sw0g/45eae33637ca92f85dd9f4a3a218e1ec/iota1_4_3.pdf

Proof of Stake. (n.d.). Retrieved from Wikipedia: https://en.wikipedia.org/wiki/Proof_of_stake

Proof of Work. (n.d.). Retrieved from Wikipedia: https://en.wikipedia.org/wiki/Proof_of_work

Schnorr, C. P. (1991). Efficient signature generation by smart cards. *Journal of Cryptology, Vol. 4, no.3*, 161-174.

Shard (database architecture). (n.d.). Retrieved from Wikipedia: https://en.wikipedia.org/wiki/Shard_(database_architecture)

Visa acceptance for retailers - Security and reliability. (n.d.). Retrieved from visa.com: https://usa.visa.com/run-your-business/small-business-tools/retail.html

Wood, G. (2016). *Ethereum: A Secure Decentralised Generalised Transaction Ledger.* Retrieved from http://gavwood.com/paper.pdf

Zamani, M., Movahedi, M., & Raykova, M. (n.d.). RapidChain: Scaling Blockchain via Full Sharding. *Proceedings of ACM SIGSAC Conference 2018*, (pp. 931-948).