Dandelion Network Whitepaper

Authors: Paul Chafe
Dr. Atefeh Mashatan
Alexander Munro
Brian Goncalves

Contributors: Duncan Cameron
Jason Xu

Dandelion Networks (dandelionnet.io)

(Dated: 25 Apr, 2022 (v.4))

TABLE OF CONTENTS

"A blockchain that claims to have solved the [Buterin] Trilemma of secure, scalable and decentralized, has either bent the laws of physics... or it has discovered a breakthrough method that solves the major blockchain scalability problems that have stumped top mathematicians and computer scientists for the past decade."

- Georgios Konstantopoulos (Konstantopoulos, 2018)

0. EXECUTIVE SUMMARY

Open blockchain networks enable global peer-to-peer transactions through the property of distributed trust (Anjum et al., 2017; Bellini et al., 2020; Karame & Capkun, 2018). Most successful blockchain models rely on either raw computing power (as in Proof of Work, PoW) or of exogenous currency (as in Proof of Stake, PoS) to commit network nodes to honest consensus. However, the accruing reinvestment cycle and economies-of-scale have driven centralization of node ownership, undermining the distributed trust property. In PoW, this has manifested in large scale mining companies making infrastructure investments on the scale of tens or hundreds of millions of dollars, excluding small operators from the network. In PoS, the store of value transfers directly from an origin currency into the staked blockchain without the need for physical infrastructure, which enables even faster centralization. A related problem is that the blockchain paradigm is inherently serial, which means scaling requires awkward work-arounds. These realities both increase costs and limit applications. We describe Dandelion, an alternative, centralization-resistant network, demonstrating long-term security, rapid finalization, and unrestricted scalability. Development to date confirms that this network can dramatically reduce gas (operating costs), finalization time, and attack vulnerabilities, while allowing a fully scalable transaction system and a wide variety of Decentralized Finance (DeFi) systems built on top of them.

1. BACKGROUND

Sufficiently decentralized networks have the property of distributed trust (Beck et al., 2016; Dorri et al., 2017; Veloso et al., 2019). This means the network may be trusted to execute transactions honestly, even if no individual on it may be. This automated trust disintermediates traditional third-party trust-providers (e.g., banks, markets, etc.), and can operate considerably more efficiently. Bitcoin and subsequent networks like Ethereum have demonstrated global demand for this service model.

Traditionally, these networks have operated under a parallel computing model which equilibrated a trade-off between three core attributes: (i) the security and integrity of the network, (ii) its intended decentralization and trust distribution, and (iii) its overall scalability. This prompted designs in which a "choose-2" trade-off over a primary attribute evolved. First expressed by the founder and designer of the Ethereum network, Vitalik Buterin, this has become the eponymous Buterin Trilemma (Altarawneh et al., 2020).

The use of the PoW method to underpin honest consensus was developed in the Bitcoin Network (Nakamoto, 2008), which operates a computational race to find the cryptographic magic numbers required to advance the blockchain. Solving for these numbers requires substantial computing power demanding a proportionally large amount of electrical power; in effect, Bitcoin encapsulates this consumed electrical power in a verifiable token. This is akin to the energy encapsulation (as labor, fuel, and materials) in gold mining and refining, which also results in a verifiable token. This serves as the basis for the network's distributed trust property and is an innovation in econophysics as much as in computer science.

Unfortunately, the PoW model is intrinsically inefficient. The central Bank of Canada found that operating the Canadian economy on Bitcoin would require 26.3 times the entire power consumption of the country (Chapman & Wilkins, 2019) and currently that network consumes more electrical power than industrialized Sweden (Rauchs et al., 2021). Although this ratio may improve, research into energy consumption as a "Granger-cause" for GDP growth suggests that the available useful energy (in quality-indexed British thermal units) to such an economy would need to expand infeasibly to satisfy the utility promised by such PoW models (Cleveland et al., 2000). One may draw the clear conclusion that Bitcoin cannot plausibly underpin a majority of economic transactions and may continue to prove problematic in other derivative PoW networks, including Ethereum. PoS systems (Buterin & Griffith, 2017; Zamfir et al.) displace this physical investment with economic investment. Given that the investment is indirect, PoS appears more efficient, but purchasing stake substitutes primary electrical power for derivative economic power. Under current economic theory (ref. Cleveland et al., 2000), economic power follows available energy inputs into that economy. Thus, PoS's extensible utility and economic value still require at least proportional energy to maintain security properties as demand rises.

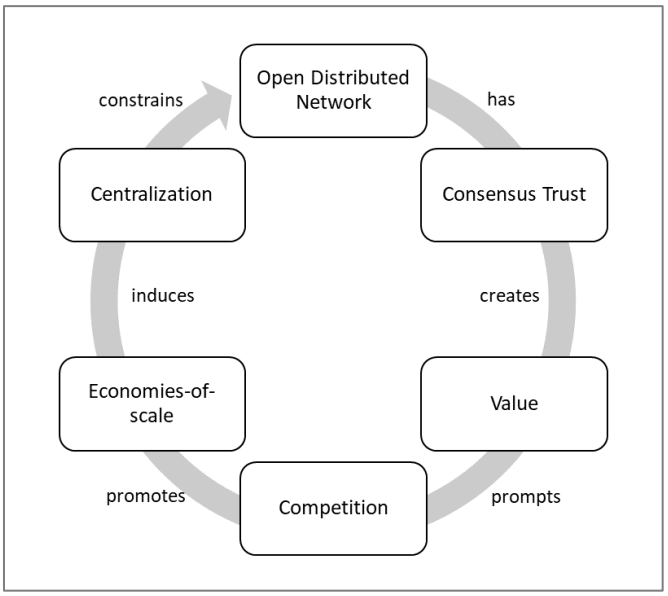


Figure 1: Qualitative feedback of blockchain's current state of art.

This energy problem is a serious concern, but more important is the second-order effect of these costs, which provide economies of scale to rent-seeking network participants through cost-reduction methods (e.g., larger mining operations) and revenue-maximization (as mining pools) (Lin William Cong et al., 2019). These have driven consolidation of network ownership, which is an expected economic process. However, in the case of open blockchain networks, centralization erodes the distributed trust property and thereby reduces the network's intended value (Swanson, 2014). Currently, just three major mining pools control over 50% of Ethereum, and just four control over 50% of Bitcoin; indeed, "51% attacks" have already been successful against smaller blockchains such as Ethereum Classic (Jenkinson, 2019) and Bitcoin Gold. Without the physical constraint of building mining infrastructure, PoS networks are likely to centralize even faster. If open blockchains are to be universally accepted as an economic medium, then not only must node-ownership be widely distributed, it must remain so in the face of centralizing forces (see Figure 1).

To prevent this, node-owners must be structurally prevented from collusive coalitions; but even absent collusion, node operators in PoW and PoS systems have available to them mechanisms to maximize their private profit at the expense of network clients (Eskandari et al., 2019).

1.1. Problem Definition

In order to provide sustainable distributed trust, a network must resolve the Buterin Trilemma (Figure 2), simultaneously providing:

(i) *Decentralization* —Network membership must be available to any node of computing power, $O(c)$[1];

(ii) *Scalability* —The entire network must have computing power $O(n)>O(c)$, where n equals network membership and $O(n)$ is its computing power, which suggests an ability to process transactions in parallel; and,

(iii) *Security* — Any malicious actor (as Adversary) must have computational power $>O(n)$ to compromise the integrity of the network.

In operation, the network must be provably secure and demonstrably decentralized in order to underpin transactions effectively, meaning scalability is the necessary sacrifice (Monte et al., 2020). Moreover, the decentralization property must hold against the pressure of economies-of-scale and other external forces. This is demonstrably not true in current systems.

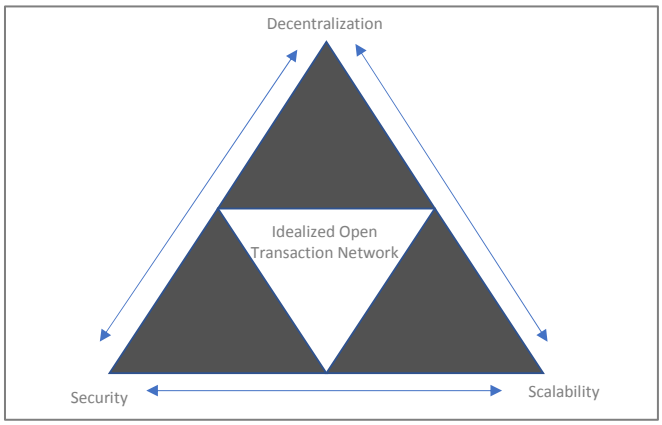


Figure 2: The "choose-2" trade-off per the Buterin Trilemma.

2. DANDELION NETWORK

We propose the Dandelion Network which can solve the Trilemma under a model which presumes its nodes and clients are (i) economically *rational* and (ii) exist *in the real world*, with the assumption set:

(i) *Senders* expect an external gain-in-trade and thus seek to clear transactions they have initiated.

(ii) *Receivers* expect to become Senders in the future (undefined) and thus seek to settle inbound transactions and to deposit proceeds into their accounts.

(iii) *Nodes* are a special subset of Receivers and are rewarded (incentivized) for processing transactions. Reward for correct and compliant processing must exceed the potential external reward for any other behavior within their choice set.

To achieve this, the Dandelion transaction network implements three key improvements to conventional blockchain technologies:

(i) *Architecture*— The Dandelion Root (Realtime Optimized Order Transactions) blocklattice is the core data model. Blockchains are inherently serial structures, whereas the blockattice is inherently parallel. This allows parallel processing of both transactions and smart contracts, increasing efficiency and throughput.

(ii) *Consensus*—the Dandelion Leaf consensus mechanism. This patented protocol offloads bandwidth requirements from nodes to clients, reducing communications complexity at the node

[1] In computer science, "Big-O" notation is typically used to relate functions tending toward limits, infinities, or asymptotes, such as computing power, storage constraints, network growth, algorithm run-time, etc. Here, we describe the limiting relationship amongst the Trilemma's determinants.

to O(1) from (typically) O(n^2). This in turn enables tens of thousands of peer-nodes the decentralization and security parameters of the network.

(iii) *Asynchronicity*—The Dandelion Stem asynchronous clear/settle paradigm decouples the clearing and settlement mechanisms, and allows cross-shard transactions. This allows the network to scale without limit.

These technologies are mutually enabling. Collectively, they achieve clear-and-settle of transactions in less than one second at unmatched high network throughput and unrestricted scaling. This makes the network as efficient as is physically possible for a given level of communications/computation technology, and a chosen level of distributed trust. This efficiency translates directly into transaction costs and gas fees several orders of magnitude lower than are possible on either competing blockchain networks or conventional transaction rails.

3. ARCHITECTURE

3.1. Definitions

(i) A *clear* is an operation in which an amount is debited from an account, creating a new balance.

(ii) A *settle* is an operation in which an amount is credited to an account, creating a new balance.

(iii) A *transaction* is a complementarily matched pair of exactly one *clear* and one *settle* operation, occurring asynchronously.

(iv) A *fee* is an amount charged for a transaction. Fees are burned (see Figure 3) by including them in the debited amount in *clear* operations but not in the credited amount of the matching *settle* operation.

(v) A *chain* or *account-chain* is a set of hash-linked balances, resulting from successive *clear* and/or *settle* operations. Every transaction links to two account-chains.

(vi) A *pennyjar* is an unordered buffer associated with an *account-chain* which stores incoming transactions prior to a *settle*. The analogy is a tip jar which stores an unordered stream of incoming donations until the receiver can count and order them.

(vii) *Pennies* are incoming, unsettled transactions stored in the pennyjar.

(viii) A *pennyroll* is a collated and serialized list of pennies.

(ix) An *account* collectively describes the designated *chain* with its associated *pennyjar* and supporting metadata, identified by the public key of a public-key/secret-key keypair pair. The public-key is used to confirm that transactions on the account are valid by proving they are signed by the associated secret-key.

(x) *Tokens* are the Dandelion network's fungible currency as exchanged between accounts through transactions.[2]

(xi) A *client* is a network-connectable device which holds the private-key to an account and executing the processes required to clear and to settle transactions.

(xii) A *node* is a computer and which earns tokens through the ongoing maintenance of a dynamic subset of accounts and the verifying/authorizing of transactions involving its designated accounts.

(xiii) *Minting* is the process by which nodes are rewarded tokens for execution of their activities.

(xiv) A *shard* is a subset of nodes across which all maintain and cross-verify the same subset of accounts. The network can include an arbitrary number of shards.

(xv) The *network* is the full set of all shards.

3.2. Data structure

The Dandelion network's basic structure is not a blockchain, but a blocklattice. This differs from existing blockchain protocols in several important respects.

(i) There are many chains. Every account maintains its own account-linked transaction-chain.

(ii) There are no blocks. Each account-level transaction forms its own link in its paired transaction-chain.

(iii) Transactions are discretely matched and paired as a distributed ledger (DL). For every outgoing (cleared) transaction sent from an account-chain, there must exist a correlating incoming transaction (settled) on another account-chain. This is analogous to double-entry book-keeping.

(iv) Chains are updated through an asynchronous clear/settle protocol.

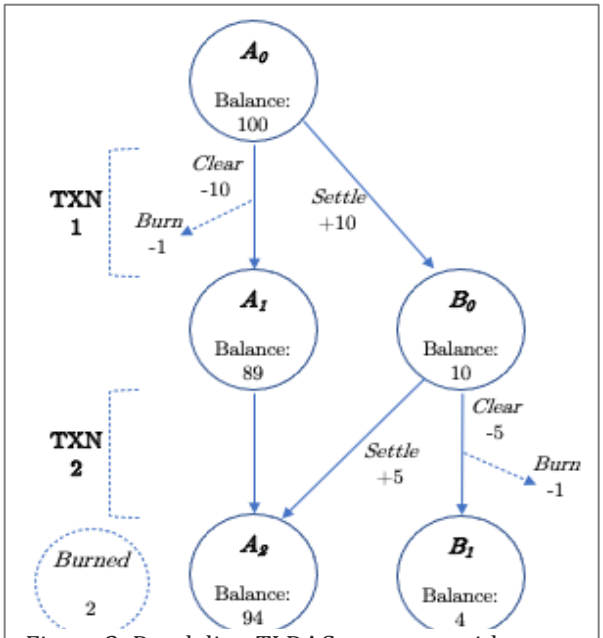


Figure 3: Dandelion TLDAG structure with state advances for a match-paired transaction, across two separate transactions.

This structure produces several important advantages over conventional blockchains:

[2] Internally, the Dandelion network's tokens are called *Pennies*; but to avoid confusion, any use of Pennies in this document strictly refers to the backlog of incoming, unsettled transactions.

(i) Each account has its own chain, thus finalization time is limited only by available bandwidth to the client's device;

(ii) Transaction order is optimized by the client (Dandelion Root).

(iii) Clearing and settling are asynchronous which allows the network to scale without limit (Dandelion Stem); and,

(iv) Transactions are processed individually and in parallel, and therefore there is no opportunity for in-block front-running, empty-block mining, or Miner Extracted Value.

3.3. State Machine

Dandelion advances state using a simple state machine to implement a two-phase commit process (see Figure 4). Every node maintains a version of this state machine for every account chain in its subset of the TLDAG. Perforated lines indicate state transitions requiring messages signed by the client's private-key. Solid lines indicate state transitions requiring the aggregated signatures of a ⅔ majority of the nodes of the account-chain's given shard.[3]

3.4. Node States

Finalized. The finalized state (State 0, S_0) is the normal "resting" state of the state machine in which all transactions have been processed up to the current transaction number (Tx_i). In this state, the account balance is available, and the client may: (i) *precommit* to a transaction; (ii) move to the *preabort* state if the client wants to abort a transaction which has been precommitted on other nodes; or (iii) advance the node to any finalized state (States 1a, 1u, or 1c) upon receiving the collated verifications confirming that the majority of its shard's nodes is advancing or has advanced its aggregate state-machines to such finalized state.

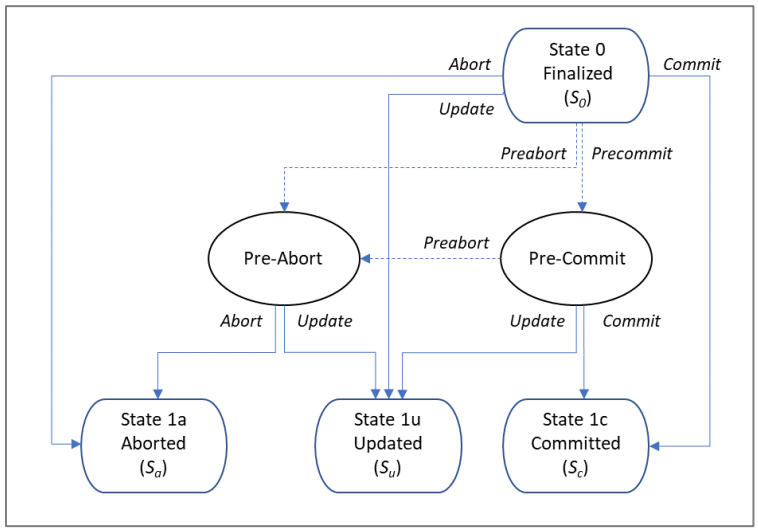


Figure 4: Dandelion's general state machine advances.

Lagging. When a node's account-chain for a given client is not current with the majority of nodes on its shard, it is *lagging*. Here, a majority of nodes has already advanced through the state machine to finalize a subsequent transaction, and the lagging node may be updated (State 1u) through an authenticated *update* or *abort* request.

Precommit. The *precommit* state indicates that the client has requested a transaction of a node and the node has validated the transaction; but the node has not yet authenticated that the client has duplicated the request across a majority of nodes on its shard. An account on a node in *precommit* is locked to new transactions. It may be moved to a committed state (State 1c) through an authenticated *commit* request; the same holds for achieving a subsequent *updated* state (State 1u) through an authenticated *update* request (if it is lagging on one or more finalizations), or to a

[3] In balancing the tradeoff between liveliness and safety when contending with byzantine failures, as a deterministic system, Dandelion uses 3f+1 which derives to the ">2/3 majority" consensus threshold.

subsequent aborted state through a *preabort* request from the client, followed by an authenticated *abort* request (State 1a).

Preabort. The *preabort* state indicates that the client is cancelling an already initiated transaction. As with the *precommit* state, the *preabort* state is locked to new transactions, but may be updated with an authenticated update request if it is lagging (advance to State 1u). An authenticated *abort* request moves the state machine to a finalized state in which the transaction number advances but the balance is unchanged (State 1a).

3.5. Shard States

Shard state. Refers to the aggregated state of all node states for a given account. A client may not be able to contact some or all of a shard's nodes at any given time. Thus, different nodes may be in different states with respect to the client's account. Shards can be in one of two shard states:

Consistent. An account is in a consistent state when a simple-majority of the nodes in its shard are either in the same state or can be moved to the same state by completing a single state transition; or,

Inconsistent. An account is in an inconsistent state when it is not in a consistent state. An inconsistent state may always be brought to a consistent state by transiting the abort process. Inconsistent states are not expected in normal operations but may arise through Byzantine failure.

4. CONSENSUS

4.1. Client-Leader

Two-phase commit is a form of leader-based consensus system, in which a single machine coordinates the advance of the state machine across all nodes. Conventional leader-based systems suffer three flaws. First, the leader is in a privileged position to determine unilaterally the unique order of transactions. This is a significant problem for markets in which either the transaction order has priority consequence (e.g., time-fluctuating prices), or where the leader may wish to prevent certain transactions from proceeding at all (i.e., transaction censorship). Second, the leader represents a single point of failure. A denial-of-service attack need only target the network's leader, a task orders of magnitude easier than attacking a majority of nodes in any large network. Most leader-based models have crash-recovery/leader-election systems to replace a failed leader rapidly, but a malicious node will be able to identify the new leader immediately as soon as the recovery process completes, and so can redirect the attack to disrupt the network continuously. Finally, the leader's communications load rises quadratically ($O(n^2)$) with the size of the network, which strictly limits both the number of nodes and the total volume of processable transactions.

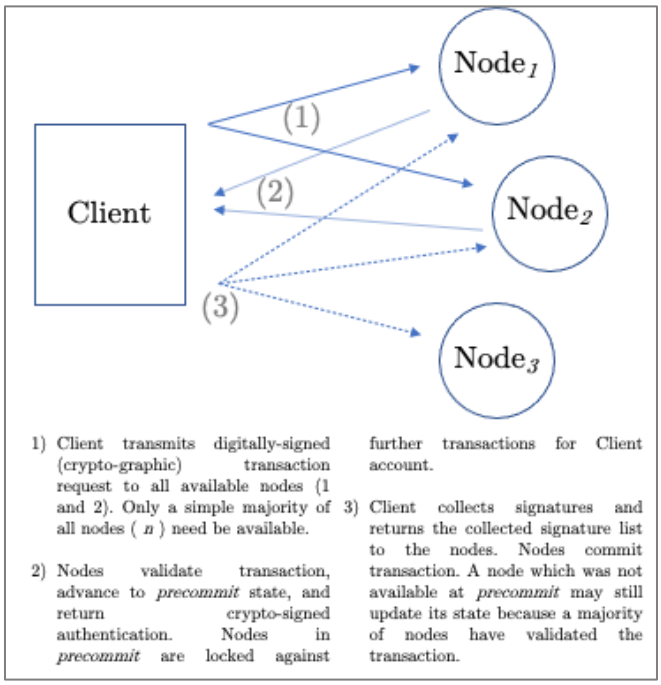


Figure 5: Client-Leader overview

Dandelion's Leaf mechanism (see Figure 5) resolves these problems using the patented client-leader paradigm. With the Client's account replicated across all nodes of its associated

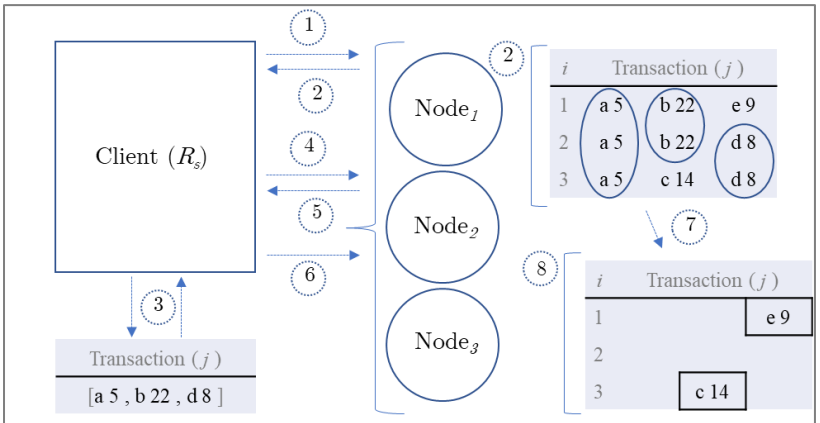
shard, the client becomes the leader for its own account-chain and is responsible for advancing the state machine across its shard. This offers several benefits. First, nodes do not and need not communicate with each other. All communications go through the client. This offloads the network's quadratic communications ($O(n^2)$) to the client, while the nodes' communication load rises only as $O(n)$ with node count, enabling a very high node count and thus a very high level of distributed trust. By definition, clients process fewer transactions than the network broadly; thus, a quadratic rise in communication load at the client does not limit total network throughput. Further, each client is free to allocate resources to the system as best befits its needs, which enables many different client configurations, including direct clear-settle, gateway access, single-device and distributed-client configurations. Finally, short of complete network shutdown, leader-targeted denial-of-service attacks can only disrupt transactions on individual accounts and only until the client identifies the attack and connects via an alternative path to its nodes.

The quadratic rise in communication load can be further improved in Dandelion's case, because the load consists of n signatures which must be received from and then retransmitted to n nodes. However an aggregable signature scheme such as Boneh-Lynn-Shachem (BLS) allows the aggregation of multiple signatures into a single signature data block. In this case, the transmission load, at the client, rises only as $O(n)$. At the node, the communications load is static (ie, $O(1)$) regardless of the number of nodes.

5. TRANSACTION

5.1. Asynchronous Clear-and-Settle

Dandelion Stem prevents double-spending by locking the sender's state machine to new transactions whilst it is in a transitional *clearing* state, thus allowing only one outbound transaction at a time. However, the receiver's account-chain may receive any number of inbound transactions at once, in any order, from different subsets of nodes in different shards. Dandelion resolves this complexity by associating each client account with an unordered data buffer (i.e., "pennyjar") which receives and stores incoming transactions for *settlement* (Figure 6). Upon finalization of a clear on the corresponding sender's account, the resulting penny is transferred into the receiver's pennyjar. Under Dandelion's Client-Leader protocol, onus then lies with the receiver client to settle transactions onto its own account-chain.



1) Receiver client ("client") of a given shard (R_s) transmits crypto-signed pennyjar request to at least one of the available nodes on its shard ($\{N_{s,i}\}$). Only a simple-majority of nodes is needed for any given penny.
2) Each node returns its full set ($\{ N_{s,i} \times R_{s,j} \}$) of pennyjar entries to client. Some nodes may not have entries for all transactions.
3) Client collates, serializes, then signs all pennies returned by the majority of nodes.
4) Client broadcasts the pennyroll.
5) Nodes verify the pennies and return a signed verification to the client.
6) Client broadcasts to nodes the verifications of the pennyroll.
7) If verified, nodes delete their entries for majority-signed transactions from the client's pennyjar.
8) Nodes process client's account-chain and hash new and ordered transaction series to the client's account-chain. Pennies for which no majority node-signatures exist remain until further state machine advances across the nodes on client's shard.

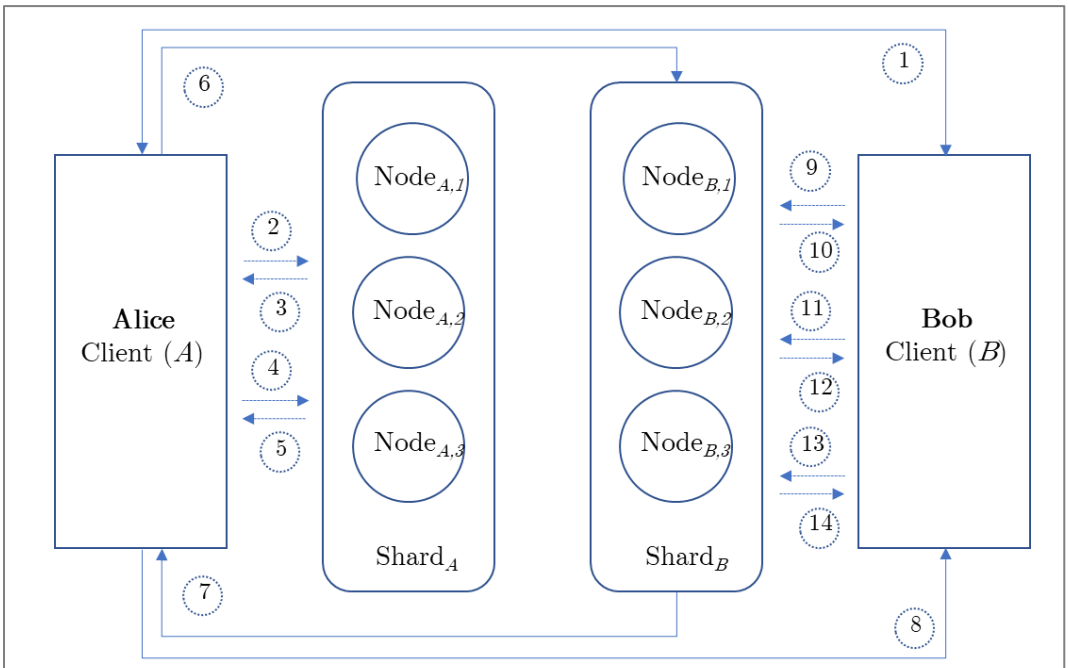
Figure 6: Dandelion's asynchronous clear-settle mechanism.

This offers the following advantages:

(i) Sender and receiver do not have to be connected at the same time to execute the transaction.

(ii) Transactions can clear and settle across shards, allowing the network to scale unrestricted.

(iii) Senders and receivers are able to negotiate and to confirm the transaction out-of-band, using any digital or physical channel as may befit the application layer (e.g., NFC, QR code, URL, post mail, etc.).



1) Alice (Sender, "A") and Bob (Receiver, "B") handshake, then exchange public keys, shard addresses, and the amount due from A.

— *CLEARING* —

2) A approves the transaction and sends transaction request to her Shard-A nodes.
3) Shard-A nodes validate the transaction, enter a precommit state and send verifications to A.
4) A collates precommit verifications and sends collation to Shard-A nodes (*Clearing* consensus is achieved).
5) Shard-A nodes commit transaction and send clearing verifications to A.
6) A collates clearing verifications and sends collated list to Shard-B nodes.
7) Shard-B nodes confirm clearance receipt to A and add transaction details to B's pennyjar.
8) A confirms to B that transaction has Cleared.

— *SETTLEMENT* —

9) B requests his pennyjar from his nodes on Shard-B.
10) Shard-B nodes return pennyjar to B.
11) B collates, serializes, then signs all pennies returned by the majority of nodes for settling order. B precommits that order to Shard-B nodes.
12) Shard-B returns precommit authorizations to B.
13) B sends the settled pennies in order to Shard-B nodes (*Settlement* consensus is established).
14) Shard-B nodes commit transaction to B's account-chain, making transaction Settled.

Figure 7: Dandelion generalized full transaction sequence.

5.2. Complete transaction sequence

Figure 7 illustrates a complete Client-Leader cross-shard transaction between sender (Alice) and receiver (Bob). Figure 7 is generalized and various permutations and configurations are readily constructed at the application layer. Importantly, under Client-Leader, clients assume responsibility for finalization. As clients will not initiate transactions which they do not intend, normal transactions will follow this generalized sequence. Reference algorithms defining sending (*1–Send*), request validations (*2–Collect States*), updating receiver account (*3–Receive*), and finally receiving (*4–Reply*), are provided in Annex A: Reference Algorithms.

5.3. Abort sequence

If a client decides not to complete a transaction which has not been precommitted on >⅔ of nodes (e.g., in the event of a disconnection or interruption), the client may choose to abort the transaction. In this case, any *precommit* nodes may be moved to the *preabort* state. On collecting >⅔ of signatures for the *preabort* state, the client may then move the transaction to the aborted state. The previously committed state is abandoned, and its transactions' pennies withdrawn from the recipient pennyjar.

Throughout the abort process, it is impossible to *clear* any of the partially completed transactions given that >⅔ of nodes are never committed to any one of them; and it is impossible to *settle* any of the partially completed transactions, as none has been cleared. Per our Assumption-1 (Senders only initiate to obtain external gain-in-trade), the preceding behavior is not economically rational and thus initiated transactions will not normally be aborted.

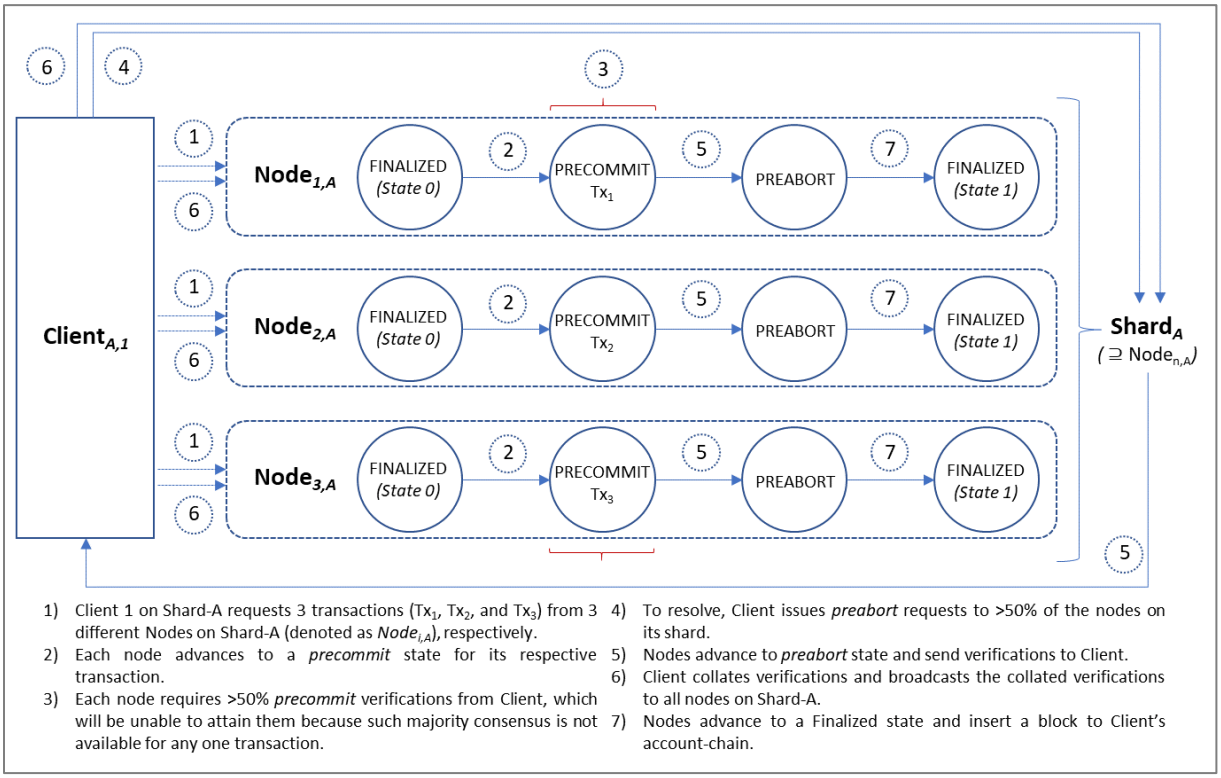


Figure 8: Inconsistent states and the Abort sequence for resolution.

Figure 8 shows the state progression through the *abort* state for a network in an inconsistent state. Note, the state number is advanced through the abort sequence, even though the balance is reverted.

5.4. Update Sequence

Nodes which are offline when a transaction is finalized will become *lagging*. To account for this, the state machine allows clients to update lagging nodes with verifications as signed by a >⅔ majority of nodes in the current finalized state. On receiving an update request, the nodes return the range of transaction IDs required to move the state machine from the furthest lagging state to the current state. The client then broadcasts these verifications to the lagging nodes. On receiving >⅔ verifications, lagging nodes can completely update the client's chain for this transaction range without transiting the *precommit-commit* process for each link in the chain.

5.5. Byzantine Clients and Inconsistent States

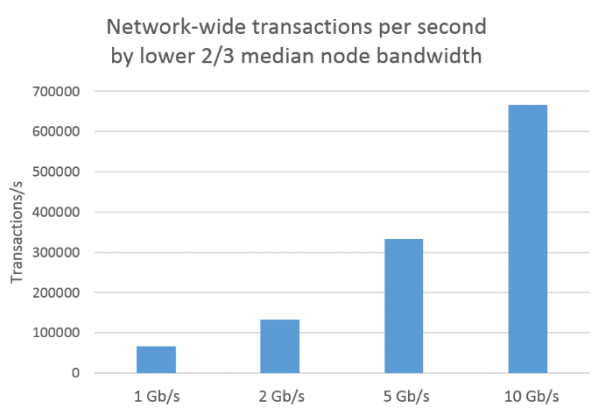
A consequence of the client leader model is that clients are completely responsible for maintaining the state of their own account chain. While it is not possible for a byzantine client to finalize more than one transaction at a time, it is possible for byzantine client to put their chain in an inconsistent state. In this case the client initiates a transaction and obtains verifications from >⅔ of the nodes, but only commits the transaction for <⅔ of nodes; it may then initiate a second transaction, maximally gaining the remaining minority verifications (<⅓), which cannot be used to finalize the second transaction on any of the nodes. The client can initiate any number of transactions on a minority (<⅓) of nodes, but because the client cannot collect a ⅔ majority of verifications for such transactions, none can finalize; rather, only the first transaction can finalize, for which >⅔ have committed.

When the byzantine failure of the responsible client is cleared, the chain can be returned to a consistent state by moving the precommitted nodes through the abort cycle. Through this process no transaction can be cleared. The account chain is advanced to the next state, with balance and transaction status equal to the previous state.

5.6. Performance

Unlike conventional blockchains, Dandelion does not have an inherent transaction rate. Rather, transaction performance parameters are governed by the number of signatures per second that can be processed and transmitted. As these figures will vary across both nodes and clients, performance will also vary both between transactions and over network life. Upgrade incentives built into the tokenomics will provide for continuous improvement as the underlying technology advances. Note that total network throughput is governed by the performance of the median node; as transactions require only strictly >⅔ consensus, clients may freely ignore slower verifications for finalization and update slower nodes later.

With this in mind, performance under present day realworld conditions can be estimated. At the node, and for the network as-a-whole, the figure-of-merit for performance is transactions per second. Using BLS aggregable signatures, signature verification costs rises linearly with network size *(O(n))*, while transmission costs scale with total throughput. This implies that capacity for both should be matched for optimal efficiency (ie, bandwidth and processing power should scale together as the network becomes more distributed, and the transaction demand rises). Nodes are rewarded for participating in consensus and so are motivated to improve their performance to ensure their inclusion in settlement rounds. Figure 9 presents the estimated transaction throughput by network size and data-rate. At a shard size of 10000 nodes, with server-grade hardware and Gbit connectivity and matching processor power, the network can process more than 250,000 transactions per second, per shard. For comparison, the VISA network averages about 1,700 transactions per second (Li, 2019) and has a peak capacity of at 54,000 transactions per second. 10Gb is commonly available for wired/fibre connections around the planet.



At the client, the figure-of-merit is finalization time. For this metric, the verification load at the client is fixed and light, but transmission demand rises as the linearly with network size. This is a concern as simple transactions between locally connected mobile devices are a major use case and. Under current technology, mobile devices have best-possible data-rate of approximately 1 Gbit/s on high quality 5G. Figure 10 presents the finalization time as a function of network size. From this analysis, it can be seen that good 5G connectivity allows direct peer-to-peer transaction sub-second finalization on a network of 16000 nodes. For a global network, this means total transaction time will be dominated by the maximum normal-condition round trip time to the most distant node, which can be taken to be a maximum of 1.5s (three round trips with 250ms trans-globe latency). However, even on a low quality 5G connection or older generation connections, transactions are finalized within seconds. For clients with slower connections, Dandelion's structure allows the use of gateways: These are systems configured to receive verifications on a low-speed channel once, and then rebroadcast those to the shard over a high bandwidth channel. For example, a client point-of-sale terminal configured for transaction settling over a fiber-link may provide a counterparty client on a mobile device with a high-speed connection to clear the transaction without any use of wireless connectivity. Performance enhancements beyond the simple peer-to-peer single-transaction model are discussed below.

Fig
siz

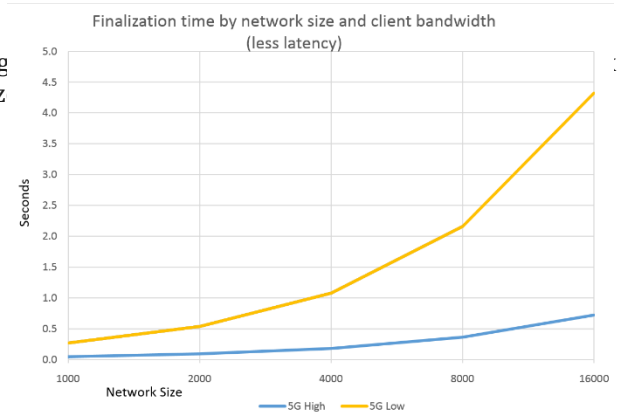


Figure 10: Finalization time by network size and bandwidth.

5.7. Performance Enhancement

Beyond these raw performance metrics, the asynchronous client-leader paradigm enables several special case situations where performance can be increased.

5.7.1. *Transaction Stacking*.

The sequence of signature exchanges as detailed above is required to complete a single transaction. However it is possible to stack any number of subtransactions into the same signature exchange. On the sending client side, the client may package any number of transaction, with different amounts and recipients into the same signature exchange. The nodes simply confirm that the sum of all the subtransactions is within the balance of the sending client. The state machine is advanced normally, and on finalization, each subtransaction is sent to its respective receiving client. On the receiving client side, the entire contents of the penny jar can be downloaded by the receiving client, a single unique order of inbound transactions for which a majority of signatures has been received is established, and that ordering is sent back and finalized across all the nodes. This is useful for high-volume applications, such as exchanges and payment systems, where transactions can be efficiently batched in short intervals.

5.7.2. *Transaction Delegation*

Both the clear and settle halves of a transaction are authorized by the cryptographic signature of the respective client. In general, the network is agnostic as to the device that

actually carries out the signature exchange sequence. In a use-case such as tap-and-pay, this allows a performance increase based on the fact that the receiving clients can generally be expected to have much higher band-width and processing power available than the sending client. In this case, the sending client may cryptographically sign the initial transaction request, and then send it locally to the receiving client. The receiving client can then finalize both the clear and settle halves of the transaction using its higher bandwidth.

The reverse is also true, and this can be combined with transaction stacking. A receiving client may download and order its entire penny jar, using transaction stacking, and then delegate the sending client (or generally any device) to perform the required transmission rounds. This technique will avoid congestion issues in high usage/high density areas where many clients use shared bandwidth, such as wifi and cellular data.

5.7.3. *Client Configuration*

A client is any device which holds a secret key associated with an account, together with the required software to carry out the Dandelion Protocol. A simple peer to peer model is two smartphones, each configured as above, which connect over local NFC to exchange transaction data, and the perform the *clear* and *settle* halves of the transaction with the nodes of the network through cellular connectivity. However the ability to stack transactions and delegate finalization allow many other configurations. A smart-card with NFC and on-board processing can authorize a transaction with an on-board secret key, and then delegate finalization to a independent point-of-sale terminal with a hardwired connection. A financial institution datacenter may hold many accounts internally, and stack transactions as required to serve its customers, before batching them to a finalization system that communicates with the network. A retail organization can have many point-of-sale terminals send transaction data over a virtual private network to a central finalizing system, while simultaneously pushing the data into their internal databases, which can then generate private intelligence in realtime. The flexibility of the client-leader configuration allows specific users to customize their performance and operating characteristics to fit their particular use case.

6. **SMART CONTRACTS**

Smart contracts are independent pieces of code running on a network with the property of distributed trust. As long as the distributed trust property holds and the network continues to operate, a smart contract will function according to its logic. Smart contracts have transactions as inputs and outputs, and the network charges fees for their execution. As a fully parallel, asynchronous network, the Dandelion network will run smart contracts with the same efficiency benefits it brings to simple transactions. Specific characteristics of the Dandelion Smart Contract platform are:

(i) *Dataflow operation*. As Dandelion is inherently parallel, it is possible to abandon the control-flow paradigm and use the dataflow model instead. This allows fully parallel smart contract operation without the risk of deadlocks or livelocks. Parallel operation represents a considerable efficiency gain over conventional blockchain systems.

(ii) *Fair ordering*. In many cases the order of transaction processing is important (eg, differing orders of buy or sell option execution can have differing financial outcomes). However Dandelion's smart contract design fairly orders incoming transactions and disambiguates

"simultaneous" submissions fairly, and without the ability for clients or nodes to predict in advance which

(iii) *No latency arbitrage*. Latency arbitrage depends on physical proximity to the exchange floor in order to receive information and execute operations in a time cycle shorter than operations farther from the floor can manage due to intrinsic communications delays. Overall network operation combined with fair ordering renders this impossible on the Dandelion Network.

7. PROVABLE SECURITY

Dandelion's security has been proven mathematically in *On the Security of the Dandelion Protocol* (Goncalves and Mashatan 2022) published in *Mathematics* 2022, (Recent Advances in Security, Privacy, and Applied Cryptography) *10*(7), 1054; - 25 Mar 2022 The full analysis is included as Annex B to this document.

8. CONCLUSION

The Dandelion network provides a resolution to the Buterin trilemma by: (i) employing a Transaction Linked Directed Acyclic Graph rather than a traditional blockchain, (ii) empowering a Client-Leader consensus mechanism which both retains sufficiency of trust properties while offloading the bandwidth demands onto the participating clients, and (iii) decoupling the clearing and settling of transactions such that asynchronous requests and approvals are intrinsically managed. Moreover, these characteristics are built (iv) using a post-quantum cryptographic model. Collectively, Dandelion enables a transaction method with real-time finalization and with highly scalable transaction throughput.

Dandelion will provide a superior mobile P2P network for transacting application layers for which existing distributed application operators may dramatically offset the risks and costs posed by finalization fees (gas costs), network seizure and centralization (consensus risk), and frontrunning and collusive activities (transaction vulnerabilities).

9. ACKNOWLEDGEMENTS

The authors would like to acknowledge the ongoing support of Dr. Peter Gregson, Dalhousie University, without whom this project would never have been possible.

10. BIBLIOGRAPHY

Altarawneh, A., Herschberg, T., Medury, S., Kandah, F., & Skjellum, A. (2020). Buterin's Scalability Trilemma viewed through a State-change-based Classification for Common Consensus Algorithms. *2020 10th Annual Computing and Communication Workshop and Conference (CCWC)*, 0727–0736. https://doi.org/10.1109/CCWC47524.2020.9031204

Anjum, A., Sporny, M., & Sill, A. (2017). Blockchain Standards for Compliance and Trust. *IEEE Cloud Computing*, *4*(4), 84–90. https://doi.org/10.1109/MCC.2017.3791019

Beck, R., Czepluch, J. S., Lollike, N., & Malone, S. (2016). *Blockchain – The Gateway to Trust-Free Cryptographic Transactions*. 15.

Bellini, E., Iraqi, Y., & Damiani, E. (2020). Blockchain-Based Distributed Trust and Reputation Management Systems: A Survey. *IEEE Access*, *8*, 21127–21151. https://doi.org/10.1109/ACCESS.2020.2969820

Buterin, V., & Griffith, V. (2017). *Casper the Friendly Finality Gadget* (p. 10). Ethereum Foundation. https://allquantor.at/blockchainbib/pdf/buterin2017casper.pdf

Chapman, J., & Wilkins, C. A. (2019). *Crypto 'Money': Perspective of a Couple of Canadian Central Bankers*. 33.

Cleveland, C. J., Kaufmann, R. K., & Stern, D. I. (2000). Aggregation and the role of energy in the economy. *Ecological Economics*, *32*(2), 301–317. https://doi.org/10.1016/S0921-8009(99)00113-5

Dorri, A., Kanhere, S. S., & Jurdak, R. (2017). Towards an Optimized BlockChain for IoT. *Proceedings of the Second International Conference on Internet-of-Things Design and Implementation*, 173–178. https://doi.org/10.1145/3054977.3055003

Easttom, W. (2021). *Modern Cryptography: Applied Mathematics for Encryption and Information Security*.

Eskandari, S., Moosavi, S., & Clark, J. (2019). SoK: Transparent Dishonesty: front-running attacks on Blockchain. *ArXiv:1902.05164 [Cs]*. http://arxiv.org/abs/1902.05164

Goncalves, B., & Mashatan, A. (2022). On the Security of Dandelion. -.

Jenkinson, G. (2019, January 10). Ethereum Classic 51% Attack—The Reality of Proof-of-Work. *CoinTelegraph*. https://cointelegraph.com/news/ethereum-classic-51-attack-the-reality-of-proof-of-work

Karame, G., & Capkun, S. (2018). Blockchain Security and Privacy. *IEEE Security & Privacy*, *16*(4), 11–12. https://doi.org/10.1109/MSP.2018.3111241

Konstantopoulos, G. (2018, January 23). *Scalability Tradeoffs: Why "The Ethereum Killer" Hasn't Arrived Yet*. https://medium.com/loom-network/scalability-tradeoffs-why-the-ethereum-killer-hasnt-arrived-yet-8f60a88e46c0

Li, K. (2019, January 30). *The Blockchain Scalability Problem & the Race for Visa-Like Transaction Speed*. https://towardsdatascience.com/the-blockchain-scalability-problem-the-race-for-visa-like-transaction-speed-5cce48f9d44

Lin William Cong, He, Z., & Li, J. (2019). (NBER Working Paper Series, p. 55). National Bureau of Economic Research. https://www.nber.org/system/files/working_papers/w25592/w25592.pdf

Mavroeidis, V., Vishi, K., D., M., & Jøsang, A. (2018). The Impact of Quantum Computing on Present Cryptography. *International Journal of Advanced Computer Science and Applications*, *9*(3). https://doi.org/10.14569/IJACSA.2018.090354

Monte, G. D., Pennino, D., & Pizzonia, M. (2020). Scaling Blockchains without Giving up Decentralization and Security: A Solution to the Blockchain Scalability Trilemma. *Proceedings of the 3rd Workshop on Cryptocurrencies and Blockchains for Distributed Systems*, 71–76. https://doi.org/10.1145/3410699.3413800

Nakamoto, S. (2008). *Bitcoin: A Peer-to-Peer Electronic Cash System*. 9.

Rauchs, M., Blandin, A., Dek, A., & Wu, Y. (2021, May 6). University of Cambridge, Judge School of Business, Centre for Alternative Finance: Cambridge Bitcoin Electricity Consumption Index—Country Ranking. *University of Cambridge - Judge School of Business, Centre for Alternative*

Finance: Cambridge Bitcoin Electricity Consumption Index. https://cbeci.org/cbeci/comparisons

Swanson, T. (2014). *Bitcoin Hurdles: The Public Goods Costs of Securing a Decentralized Seigniorage Network which Incentivizes Alternatives and Centralization*. http://www.ofnumbers.com/wp-content/uploads/2014/04/Bitcoins-Public-Goods-hurdles.pdf

Veloso, B., Leal, F., Malheiro, B., & Moreira, F. (2019). Distributed Trust & Reputation Models using Blockchain Technologies for Tourism Crowdsourcing Platforms. *Procedia Computer Science*, *160*, 457–460. https://doi.org/10.1016/j.procs.2019.11.065

Vijayalakshmi, P. R., & Raja, K. B. (2012). Performance analysis of RSA and ECC in identity-based authenticated new multiparty key agreement protocol. *2012 International Conference on Computing, Communication and Applications*, 1–5. https://doi.org/10.1109/ICCCA.2012.6179168

Zamfir, V., Rush, N., Asgaonkar, A., & Piliouras, G. (n.d.). *Introducing the "Minimal CBC Casper" Family of Consensus Protocols*. 29.

Zinsmeister, N., & Robinson, D. (n.d.). *Hayden Adams hayden@uniswap.org*. 10.

ANNEX A: REFERENCE ALGORITHMS

Algorithm 1: SENDTX (SenderAccount, TxNumber, DestAccount, Amount)

```
Tx→ (SenderAccount ←SenderAccount, TxNumber ←TxNumber, DestAccount
←DestAccount,
 Amount ←Amount)
Message→ (Tx ←Tx, Type ←SENDTX σ ←σ(Message, SenderAccount→ K_S)
(N_i , N_f ) ←SenderAccount→ Shard (N_i , N_f )
for n ∈ (N_i ...N_f ) do
  TRANSMIT (n, Message)
Message→ Verifications ←COLLECTSTATE(SenderAccount)
if #{v ∈ Message→ Verifications | Σ(v, v→ σ, N_v→ K_p ∧ v → State = PRECOMMIT ∧ v→
Tx=Tx} >
 2(N_f – N_i)/3 then
  Message→ Type ←CLEARTX
  Message→ σ ←σ(Message, SenderAccount→ K_s)
  for n ∈ (N_i ...N_f ) do
    TRANSMIT (n, Message)
if SenderAccount→ Shard ≠ AccountList[DestAccount]→ Shard then
  Message→ Verifications ←COLLECTSTATE (SenderAccount)
  if #{v ∈ Message→ Verifications | Σ(v, v→ σ, N_v→ K_p) ∧ v→ State = FINALIZED ∧ v →
   Tx=Tx} > 2(N_f – N_i)/3 then
    (N_i , N_f ) ←AccountList[DestAccount]→ Shard
    Message→ Type ←CLEARXSHARDTX
    Message→ σ ←σ(Message, SenderAccount→ K_s)
    for n ∈ (N_i ...N_f ) do
      TRANSMIT (n, Message)
```

Algorithm 2: StateList COLLECTSTATE (Account)

```
(N_i , N_f ) ←Account→ Shard(N_i , N_f )
while j < N_f – N_i ∧ ~TIMEOUT(Timeout) do
  if Σ(Reply ←WAITRECEIVE(),Reply→ σ, NodeList[Reply→ NodeAccount]→ K_p) then
    StateList ←StateList⌢Reply→State
    j++
return StateList
```

Algorithm 3: RECEIVETX (TxNumber,DestAccount)

```
Tx→ (TxNumber ←TxNumber, DestAccount ←DestAccount)
Message→ (Type ←REQUESTPJ, Account ←DestAccount)
(N_i , N_f ) ←DestAccount→ Shard(N_i , N_f )
for n ∈ (N_i ...N_f ) do
  TRANSMIT (n, Message)
while j ≤ (N_f – N_i) ∧ ~ TIMEOUT(Timeout) do
  PennyJarList→ PennyJarList⌢WAITRECEIVE()
  j++
Tx→ Pennies ←{p ∈ PJ ∈ PennyJarList | #{pj ∈ PennyJarList | p ∃ pj} > 2(N_f – N_i)/3 }
```

```
Message→ Type ←CONFIRMTX
for n ∈ (N_i ...N_f ) do
  TRANSMIT (n, Message)
Message→ Verifications ←COLLECTSTATE(SenderAccount)
if #{v ∈ Message→ Verifications | Σ(v, v→ σ, N_v→ K_p) ∧ v→ State = PRECOMMIT ∧ v →
 Tx=Tx} > 2(N_f − N_i)/3 then
  Message→ Type ←SETTLETX
  Message→ σ ←σ(Message, SenderAccount→ K_s)
  for n ∈ (N_i ...N_f ) do
    TRANSMIT (n, Message)
```

Algorithm 4: Reply NODE(Message)

```
Tx ←Reply→ Tx ←Message→ Tx
Reply→ NodeAccount ←Node→ NodeAccount
(N_i , N_f ) ←Node→ Shard(N_i , N_f )
if (∃ AccountList[Tx→ SenderAccount] ∧ Σ(Message, Message→ σ, AccountList[Tx→
SenderAccount]→ K_p)) ∨ (∃ AccountList[Tx→ SenderAccount] ∧ Σ(Message, Message→ σ,
AccountList[Tx→ DestAccount]→ K_p))
  then
    switch Message→ Type do
      case SENDTX do
        if AccountList[Tx→ SenderAccount]→ State = FINALIZED ∧ Tx→ Amount+Fee
        < AccountList[Tx→ SenderAccount]→ Chain[#Chain]→ Balance ∧ Tx→ TxNumber
=
        #Chain then
          Reply→ (State ←AccountList[Tx→ SenderAccount] →State ←PRECOMMIT), σ
←σ
          (Tx, NodeAccount→ K_s)

      case CLEARTX do
        if AccountList[Tx→ SenderAccount]→ State ≠ PREABORT ∧ Tx→ TxNumber =
#Chain
          ∧ (∀v ∈ Message→ Verifications ∃ NodeList[v→ NodeAccount] | Σ(v→ Tx, v→ σ,
          NodeList[v→ NodeAccount]→ K_p)= ⊤ ∧ ∀ w ∈ Message→ Verifications ∃ w→ Tx =
v→
          Tx) ∧ #{Messages→ Verifications} > 2(N_f − N_i)/3 then
          TxHash ←SHA(AccountList[Tx→ SenderAccount]→ Chain[TxNumber]→ Hash,
Tx)
          AccountList[Tx→ SenderAccount]→ Chain⌢TxHash
          AccountList[Tx→ SenderAccount]→ (Balance ←AccountList[Tx→
SenderAccount]→
          Balance - Tx→ Amount - Fee, Accountlist[Tx→ SenderAccount]→ TxNumber
←Tx→
          TxNumber++)
          Penny→ σ ←σ (Penny, Node)
          if AccountList[Tx→ DestAccount]→ Shard = Node→ Shard then
            AccountList[Tx→ DestAccount]→ PennyJar ←AccountList[Tx→ DestAccount→
            PennyJar ∪ Penny
        Reply→ (State ←SenderAccount→ State ←FINALIZED, Penny ←Penny)
```

case *CLEARXSHARDTX* **do**
if $Tx \notin AccountList[Tx \rightarrow DestAccount] \rightarrow Chain) \wedge (\forall\, v \in Message \rightarrow Verifications\ \exists$
$NodeList[v \rightarrow NodeAccount] \mid \Sigma(v \rightarrow Tx, v \rightarrow \sigma, NodeList[v \rightarrow NodeAccount] \rightarrow K_p) = \top$
$\wedge\ \forall$
$w \in Message \rightarrow Verifications\ w \rightarrow Tx = v \rightarrow Tx) \wedge \#\{Message \rightarrow Verifications\} > 2(N_f -$
$N_i)/3$ **then**
AccountList[Tx→ DestAccount]→ PennyJar ←AccountList[Tx→ DestAccount]→ PennyJar ∪ {Message→ Penny}

case *CONFIRMTX* **do**
if $AccountList[Tx \rightarrow DestAccount] \rightarrow State = FINALIZED \wedge Tx \rightarrow TxNumber = \#Chain \wedge$
$AccountList[Tx \rightarrow DestAccount] \rightarrow PennyJar \mid P = p)$ **then**
Reply→ (State ←AccountList[Tx→ SenderAccount]→ State ←PRECOMMIT), $\sigma \leftarrow \sigma$
(Tx, NodeAccount→ K_s)

case *SETTLETX* **do**
if $AccountList[Tx \rightarrow DestAccount] \rightarrow State \neq PREABORT \wedge Tx \rightarrow TxNumber = \#Chain$
$\wedge\ (\forall\, v \in Message \rightarrow Verifications)\ \exists\ NodeList[v \rightarrow NodeAccount] \mid \Sigma(v \rightarrow Tx, v \rightarrow \sigma,$
$NodeList[v \rightarrow NodeAccount] \rightarrow K_p) = \top \wedge \forall\, w \in Message \rightarrow Verifications\ w \rightarrow Tx = v \rightarrow$
$Tx) \wedge \#\{Messages \rightarrow Verifications\} > 2(N_f - N_i)/3$ **then**
AccountList[Tx→ DestAccount]→ PennyJar ←{AccountList[Tx→ DestAccount]→ PennyJar | ∀ p ∈ AccountList[Tx→ DestAccount]→ PennyJar ∉ Tx→ Pennies}
AccountList[Tx→ DestAccount]→ Chain⌒SHA(AccountList[Tx→ DestAccount]→ Chain, Tx)
AccountList[Tx→ DestAccount]→ (Balance ←AccountList[Tx→ DestAccount]→ Balance + $\sum_{t=0}^{|Tx \rightarrow Pennies|}$ Tx→Pennies[i]→ Amount, Accountlist[Tx→ DestAccount]→
TxNumber ←Tx→ TxNumber + #{Tx→ Pennies})

case *REQUESTPJ* **do**
Reply→ PennyJar ←AccountList[DestAccount]→ PennyJar
REPLY(Reply)

ANNEX B: ON THE SECURITY OF THE DANDELION PROTOCOL

Paper: *On the Security of the Dandelion Protocol. Goncalves and Mashatan, 2022*

Journal: *Mathematics*
Special Issue: Recent Advances in Security, Privacy, and Applied Cryptography

DOI: https://doi.org/10.3390/math10071054

Original: https://www.mdpi.com/2227-7390/10/7/1054