WORK IN PROGRESS

Proof of Real-time Transfer: A Consensus Protocol for Decentralized Data Exchange and Storage

Abstract

Building a decentralized data platform that guarantees data security, data integrity and data privacy while catering to the real-time storage and retrieval needs of applications will transcend today's digital economy with new opportunities for innovation and growth across sectors. The challenges reside in being able to handle the sheer volume, variability and velocity that general purpose data transactions are known for and delivering on the Web3 promises of greater security, transparency and accountability.

We are building Kandola, a decentralized data exchange and storage platform, that shall cater to the storage and retrievel needs of IoT, healthcare, social and gaming. The platform is driven by a new consensus protocol named *Proof of Real-time Transfer* (PoRT, in short). This paper details the design considerations and inner workings of PoRT consensus and motivates the protocol's readiness for Byzantine Fault Tolerance. PoRT's security gurantees stem from fusion of cryptographic primitives with game theoretic constructs and is built to meet the demands applications shall place on scale, latency and reciprocal throughput. Consider this paper as a preview to the decentralized platform that our Team is building. Detailed performance benchmarks will be published in a subsequent paper.

Keywords: Consensus protocol, decentralized storage, batch derivatives, parallel chains

1. Introduction and Motivation

The digital economy has experienced tremendous growth in recent years, resulting in an unparalleled increase in data generation and consumption. The global data market is estimated to be valued north of $1.2 trillion, with expectations for continued expansion. This rapid escalation in data volume has created an urgent need for efficient and secure storage solutions. While centralized systems offer certain efficiencies, they have invariably been subject to security breaches, singular points of failure, high costs and under sporadic instances, data censorship. Building a holistic decentralized data platform that facilitates the exchange, storage and retrieval of structured data streams, that comes with the security and privacy guarantees of Web3 and poses high scalability can address real-time data needs of applications and unlock the hidden potentials behind decentralized data marketplaces.

Kandola is an innovative decentralized infrastructure solution that enables applications to exchange, query, and analyze structured data. Kandola facilitates the decentralized verification and on-chain storage of data, fostering the development of verifiable and privacy preserving applications. It empowers applications to define their own data schema, maintains their data on dedicated chains, ensures comprehensive security and privacy while facilitating transparent cost evaluations. Kandola is powered by a consensus protocol named Proof of Real-time Transfer (PoRT) one that is built to facilitate the decentralized exchange, storage and retrieval of general purpose data. This paper introduces PoRT protocol while simultaneously providing a system level overview of Kandola.

PoRT protocol is built for permissionless settings where adversaries can either be static or adaptive. With data encryption at source, end-point authentication, standardized (and yet customizable) application-driven message schemas and decentralized verification of data compliance, PoRT protocol guarantees the security, privacy and integrity of schema-compliant data all through its life-cycle on the platform. Assuming partially synchronous network settings, PoRT is designed to guarantee safety (always) and liveness (eventually). Given the domain served, PoRT is designed to handle high

transactions volumes and the steep demands placed on the finite physical resources of the decentralized platform.

1.1 Core Contributions

Below, the core contributions made through PoRT protocol are discussed.

- To the best of our knowledge, we are the first to deploy *two-way cryptographic sortition* in randomized task assignments. The task assigner identifies a set of potential assignees using crytographic sortition and the assignees upon receiving the task assignment invokes cryptographic sortition to either accept the assignment or reject it or reassign it another. Two-way cryptographic sortition renders grinding attacks largely ineffective and hence plays a critical role across different stages of PoRT protocol.
- We adopt parallel chains whereby chains are dedicated solely for an application for the lifetime of the application. For each chain, a leader is elected for an epoch. Verifiable random functions are used for both leader election and epoch determination. Only the elected leader can propose blocks for the respective chains. The above design prevents forking and paves way to instant block finality.
- We use *dynamic subnets*, the evolution of which is governed by two-way cryptographic sortition, for batch verification (a batch is simply an aggregation of transactions). The topology of the dynamically evolving subnet is not predictable, but is fully verifiable.
- We introduce a game theoretic construct named *batch derivatives* that serves the role of establishing that nodes that were identified for batch verification completed their tasks in full adherence to protocol. Batch derivatives render establishing out-of-band communication channels for deriving answer-keys for batch verification futile.
- In the context of decentralized storage, we construct two epoch-based static subnets addressing the state machine replication needs of transaction logs (on chain) and a CRUD friendly local database. The former is called Transaction Log State Machine (TLSM) and the latter is called Database State Machine (DBSM).
  - TLSM is highly scalable, provides verifiability in storage and facilitates storage tracking.
  - The DBSM engine, on the other hand, provides data storage guarantees, data redundancy, and availability.
  - The integrity of TLSM and DBSM is maintained through check-pointing and through audits, with economic incentives for nodes fostering safety and liveness and penalties imposed on nodes displaying malicious behavior

  This two-tier system enables the decentralization of any underlying database while retaining all the native features of databases.

2. Prior Art

The primary goal of Byzantine Fault Tolerant (BFT) protocols is guaranteeing *safety* and *liveness* in the presence of arbitrary failure under corresponding assumptions made on network delay (synchronous, or asynchronous, or partially synchronous operational settings). The cumulative size of arbitrary failure instances is capped at less than one-third of that of the network. The design choices behind state of the art BFT protocols encompass identifying a suitable network topology, one or more node-to-node communication protocols, modeling message complexity, identifying the right consensus type (deterministic or probabilistic) that best fits the design and importantly embracing cryptographic primitives that secure the protocol and impart verifiability in its execution. Some protocols periodically elect leaders for block proposals and while some other embrace leaderless designs. Sharding and randomized sampling for consensus have been popularly adopted for improved scale. (Berger et al. 2023) provides a comprehensive summary on state of the art BFT protocols.

Synchronized network settings are largely unrealistic in permissionless environments. BFT protocols are typically built for partially synchronous settings and more recently for asynchrononous settings. HotStuff (Yin et al. 2019) is a partially synchronous BFT consensus protocol that adopts dynamic leader election based on weighted voting mechanism, and achieves lowered message complexity and faster convergence than other such protocols that adopt round-robin leader election. Mir-BFT (Stathakopoulou, David, and Vukolic 2019) deploys an approach that allows multiple parallel leaders to propose blocks simultaneously upon partitioning the hash spaces into buckets of equal size and assigning leaders for each. Algorand (Gilad et al. 2017) employs VRF in the form of cryptographic sortition to randomly select participants for gathering consensus. Avalanche (Rocket et al. 2019) proposes a probabilistic leaderless consensus protocol that relies of metastability whereby nodes converge on a decision through randomized sampling of verdicts from across the network.

HoneyBadgerBFT (Miller et al. 2016) is centered around the Asynchronous Common Subset primitive whereby nodes choose a randomized subset of transactions from their respective buffers, apply threshold encryption and generate blocks comprising a union of the verified transactions. Internet computer consensus (Camenisch, Drijvers, and Hanke 2022) leverages chain-key cryptography in building a family of atomic broadcast protocols designed for safety (always) and liveness (eventually) serving different message complexities. Dumbo (Guo et al. 2020) (Guo et al. 2022) is a new BFT consensus protocol that introduces a hybrid architecture that combines the benefits of both leader-based and leaderless approaches. Two atomic broadcast protocols are proposed, named *Dumbo1* and *Dumbo2*, that have asymptotically and practically better efficiency than HoneyBadger BFT. Kauri (Neiheiser, Matos, and Rodrigues 2021) proposes a leaderless BFT architecture that leverages a pipelined tree-based approach for message dissemination and aggregation to achieve high throughput while maintaining low latency and low communication overhead

The protocols above are fundamentally built upon a variety of cryptographic primitives that secure the protocol and on occasions yield scaling advantages. While digital signatures and collision resistant hash functions are standard cryptographic primitives that are invariably adopted, threshold signatures, erasure coding, secret sharing, multi-signatures, cryptographic sortition are popularly adopted for improving security and at certain instances, improving scalability (Berger et al. 2023). Omniledger (Kokoris-Kogias et al. 2018), Elastico (Luu et al. 2016), RapidChain (Zamani, Movahedi, and Raykova 2018), Chainspace (Al-Bassam et al. 2017), Free2Shard (Rana et al. 2022) collectively discuss the raw mechanics behind state sharding, compute sharding, network sharding and byzantine tolerance in the presence of sharding for improved scalability.

Decentralized storage networks serve the purpose of secure storage and retrieval of data, with the summarized service guarantees discussed below:

- Establishing time-continuity in storage: Data should be stored continuously for the storage duration requested and sufficient copies of data should exist to handle failure instances
- Persistence in data redundancy: Decentralized networks have high churn rate by name. Multiple copies of data should exist across the network. Additionally, protocols should be designed such that existing nodes replicate data that was lost as a result of node failures or node departures and guarantee data availability.
- Storage proofs: Upon audit, nodes should be able supply verifiable proofs of data storage.
- Storage tracking and Efficient retrieval: Through distributed hash tables or other means, the protocol should enable tracking of data stored (either in whole or as fragments) and support efficient retrieval.

Filecoin (P. Labs 2017) is built atop IPFS (Interplanetary File System) (Benet 2014), a distributed file system that allows users to store and access files on peer to peer network. Filecoin establishes storage proofs and data availability through Proof of Replication and Proof of Spacetime. Crust (Crust 2020) adopts Meaningful Proof of Work and Guaranteed Proof of Stake, for serving the same purpose.

Sia (Vorick 2014) adopts erasure encoding and stores fragments of data across the network and has mechanisms in place to generate Proof of Storage of hashed data fragments. Arweave (Williams and Jones 2018) adopts Proof of Access for storage proofs and Wildfire protocol for incentivized data replication and retrieval. Swarm (Swarm 2021) and Storj (S. Labs 2018) are decentralized storage solutions that are built for Ethereum interoperability. (0xphillian and Labs 2022) provides a comprehensive account on different decentralized storage providers.

3. Proof of Real-time Transfer (PoRT) Consensus

3.1 Preliminaries

In this subsection, we shall begin with a handful of definitions and the relevant notations, and shall list the cryptographic primitives that PoRT adopts.

1. Let N be a set of n nodes $\{N_1, N_2, ..., N_n\}$. Nodes can be of type *full-node* or *storage-node*. A full-node runs the consensus protocol and plays a role in decentralized storage. The storage node plays a role only in the latter.
2. Let $S = \{s_1, s_2, ..., s_n\}$ correspond to the stakes of the respective nodes, the amount of coins or tokens that each node has invested as a collateral to participate in the network.
3. Let the number of Byzantine nodes in the network be f. PoRT protocol is designed for a composition of honest nodes and Byzantine nodes under which $n \geq 3f + 1$.
4. Let A be a set of a applications $\{A_1, A_2, ..., A_a\}$. Applications are data producers and / or consumers of data.
5. Let notation $[x]$ denote the set $\{1, 2, ..., x\}$.
6. Let H be a collision resistant hash function (SHA256).
7. Let DSA_KeyGen, DSA_Sign and DSA_Verify be the digital signature algorithms for the generation of key pairs, signing of messages and verification of signatures respectively.
8. Nodes and applications (henceforth referred to as entities) generate their own public-private key (pk_i, sk_i) using KeyGen, where $i \in [n + a]$. Their decentralized identity (DID) $id_i = \mathrm{H}(pk_i)$.
9. Let R be the Bootstrap registry. Every entity registers itself with R by submitting a DID document that comprises their public key(s), DID(s), service endpoints and other DID related metadata.
10. Given a message m and N_i's private key sk_i, signature S is generated as $S = \mathrm{DSA\_Sign}(m, sk_i)$ and the signature is verified as $\mathrm{DSA\_Verify}(S, pk_i)$.
11. Let Dx and Dh correspond to the XOR distance and hamming distance functions respectively. Let a and b be two binary strings. The XOR distance between them is $Dx(a, b) = a \oplus b$, where $\oplus$ is the XOR operator. The hamming distance between them is $Dh(a, b) = a \sqcup b$, where $\sqcup$ is the hamming distance operator.
12. Let TS_KeyGen be the distributed key generation function under the context of threshold signature. Let DSA_Sign and DSA_Verify, from above, serve their respective roles under the new context. Let t correspond to the threshold value pertaining to the minimum number of signatures required.
13. Verifiable Random Functions (VRF) are realized in PoRT as follows: Given a public input x, a node N_i generates a random number $\gamma = \mathrm{H}(\mathrm{DSA\_Sign}(x, sk_i))$ and generates a proof $\pi = \mathrm{DSA\_Sign}(x, sk_i)$. γ is a number that only N_i gets to generate from x. If supplied with (x, γ, π), other nodes can corroborate that γ was derived from x using $\mathrm{DSA\_Verify}(x, \gamma, pk_i)$ and verifying if $\gamma = \mathrm{H}(\pi)$.
14. Let P be a function that takes a 256 bit number r as input and generates a number $\hat{s}$ between $(0, 1)$: $\hat{s} = P(r)$. Let $r_1, r_2, ..., r_{32}$ be ordered set of bytes in r. A byte s is generated from r : $s = r_1 \oplus r_2 \oplus ... \oplus r_{32}$ and is normalized as $\hat{s} = s/256$
15. Let Q be a function that takes three inputs: a 256 bit number r, positive integers p and q and generates p numbers in the range $(0, q)$ by applying bitwise right shift operation (the operator

being >>). on r followed by a *modulo* q. Q generates an array of positive numbers $\{m_1, m_2, ..., m_p\}$ where $m_i = (r >> i) \bmod q$, $\forall i \in [p]$

3.2 Decentralized Identity Management

PoRT furnishes the decentralized public key infrastructure (DPKI) and other relevant DID SDKs that enable every entity that is associated with the decentralized platform to create and manage its own decentralized identity. Entities create their decentralized identities as mentioned in items 8 and 9 in the previous subsection. Decentralized identities are globally unique, cryptographically verifiable, are resolvable with high availability and plays a fundamental role in preserving the privacy of entities. Further, it provides a framework for auto authentication and supports interoperability across platforms.

The bootstrap registry R (item 9) is maintained at an identity management blockchain (such as Sovrin, uPort) and a set of nodes from the network are identified to serve as DID nodes serving as the network's liaisons for state maintenance of R and for data retrieval. Per protocol, nodes can either connect with the DID nodes to retrieve the most current DID documents of other entities or retrieve the same from immediate peers or correspond directly with the external identity management blockchain in and maintain local repositories of R.

3.3 Network Model

PoRT protocol adopts a scale-free, semi-structured, self-organizing peer-to-peer (P2P) overlay network. The desired properties of such a network are low propagation latency, efficiency in query-routing for resource discovery and resource sharing, resilience to network churn, and supporting high scalability while remaining free from hierarchical organization. The topology of the proposed network and the underlying mechanisms for data storage and retrieval, both draw inspirations from constructs adopted by structured and unstructured P2P overlay networks and optimizations proposed for each (Kademlia (Maymounkov and Mazières 2002), Perigee (Mao et al. 2020)).

3.3.1 Network Topology

In structured P2P overlay networks, nodes and data objects are organized onto a keyspace, as (*key*, *value*) pairs. Nodes identify their peers through a notion of *proximity* based on their respective keys. Further, there is a deterministic mapping between data objects and nodes identified based on key proximity. The network adopts Distributed Hash Tables (DHT) (Hassanzadeh-Nazarabadi et al. 2021) as a substrate that holds information on data object location and facilitates efficient resource sharing. Unstructured P2P overlay networks, on the other hand, organize nodes in a flat or hierarchical random graphs. Nodes identity their peers through randomized connectivity, or through a mechanism that serves their mutual interests. Protocols such as flooding, random walks and expanding-ring time-to-live search are typically used for querying content. While the former network's key-based routing is scalable and locates rare data items efficiently, the latter is better suited for networks with high churn and locates highly replicated content with much lesser overhead.

Let the proposed network be represented as an undirected graph $G = (V, E)$ with n nodes $N_i \in V, i \in [n]$, each assigned a unique 256 bit identity id_i (also the *key*) obtained from hashing their respective public keys. The set of edges $e_{ij} \in E$, where $i \in [n]$, $j \in [n]$, $i \neq j$ is set to 1 when node N_i and N_j are connected and set to 0 otherwise. They are organized into binary-tree with their positions identified with the unique prefixes to their respective identities. Nodes connect with a selected set of peers and maintain a routing table facilitates query-routing and resource localization as illustrated below.

- For each node, the tree is partitioned into successively lower sub-trees based on binary prefixes of identities that did not contain the node. Nodes look for at least one peer from within each

sub-tree.

- Distance between two nodes n_i and n_j is defined by the XOR metric (Maymounkov and Mazières 2002), $d(n_i, n_j) = id_i \bigoplus id_j$. For each $0 \leq m \leq 255$, a node looks towards maintaining a list of nodes (called $k-buckets$) whose distance is in the range $(2^m, 2^{m+1})$ from itself.
- In addition to keyspace proximity, nodes factor in network heterogeneities and node configurations (geographic distance, bandwidth, compute, memory) that directly impact broadcast latency in identifying an optimal set of peers (Mao et al. 2020).
- From within each sub-tree the $k-bucket$ lists comprise (node id, port number, IP address, latency, node up-time). The lists within each bucket is sorted based on a weighted combination of latency, node up-time and a recency of interaction factors. Nodes that responded to a request recently are weighed more.

The remote procedure calls namely, {PING, STORE, FIND_NODE, FIND_VALUE} that are defined in Kademlia are borrowed as is for PoRT.

3.3.2 Communication model

We assume partially synchronous operational settings, whereby messages sent by honest nodes will get delivered within a certain time δ after the passage of an unknown *global stabilization time*. The protocol handles delays in message transmission by incorporating timeouts and requests for re-transmissions. Time-out events can induce nodes to temporarily update their respective k-buckets until normal operational settings are observed. During instances of unpredictability in network conditions, when time-out events are unusually frequent, nodes resort to temporarily updating their k-buckets until normal operational conditions resume or establish new connections with peers using randomized connectivity and adopt practices from unstructured network settings for resource discovery.

Nodes communicate with one another generally through direct port-based communications and resort to peer-to-peer gossip sublayer for network wide dissemination of messages. PoRT protocol adopts HTTP/3 QUIC protocol as the underlying transport protocol.

3.4 Standardized Data Schema and Transaction Verification

A data transaction can correspond to any one of (a) transfer of data between two or more applications (type: create or update) (b) transfer of data to self (type: create) (c) data query (type: read) (d) data deletion (type: delete). The services requested can fall under one of three types: (a) delivery and storage (b) storage only (c) data retrieval.

Let D be a set of standardized data schemas $\{d_1, d_2, ..., d_k\}$, where each schema defines an application specific structure of the data payload, delineating the fields that make up the data header and body. For every d_i there exists a unique URI u_i that points to an entry in the bootstrap registry R. While applications are free to adopt an existing data schema or use their own customized data schema, the data fields and formats identified for data conformity should be adhered to for the platform to accept the transactions. Let (pk_p, sk_p) and (pk_c, sk_c) be the key pairs of the producer and consumer respectively. Let $[h|0b]$ correspond to the two header and body of the data transaction dt. The hash signature $(dt_s ign)$ field corresponds to the following: DSA_Sign(H([h || DSA_Sign(b, pk_c)]), sk_p). Table 1 illustrates a sample data transaction that adopts the standardized schema for IoT.

3.4.1 Data Transaction Verification Function

Let $DT = \{dt_1, dt_2, ...dt_k\}$ be a set of data transactions. The data transaction verification function be V: DT $\rightarrow \{0, 1\}$, with $V(dt) = 1$ for valid transactions and $V(dt) = 0$ otherwise. Source authenticity, signature validity, data integrity and schema compliance are primarily established in data transaction validation. Data transaction validation entails the following: [Verify if H(dt) = DSA_Verify(dt_sign,

Table 1. Standardized Data Schema

Header	
Application DID	Hash of the application's public key
Transaction ID	A GUID
Producer address	DID URI of the producer
Consumer address	DID URI of the consumer or group or self
Command type	An enum pointing to one of 4 commands: Create (insert), Read (query), Update and Delete
Nonce	An incremental index that signifies the order of data transactions from an application
Schema URI	URI adopted for the data payload
Timestamp	Dispatch timestamp
Hash Signature	Signed hash of the full data transaction
Body	
Raw Data	Actual payload, with the field values encrypted.

pk_p) ∧ verify if schema URI is valid ∧ verify raw data schema compliance ∧ verify {nonce, command type} validity ∧ verify producer authentication].

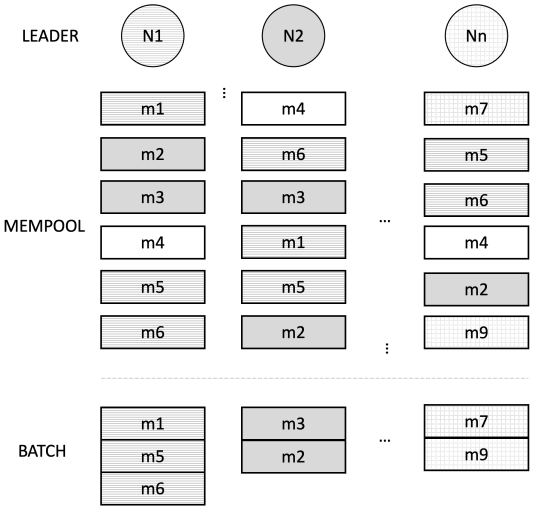

Figure 1. Messages dispatched by applications making it to mempools across the network. The node designated as leader for a chain, prepares batches comprising of messages sent by the application that is tied to the chain.

3.5 Mempool data structure and Entropy Source

Mempool is a data structure (Figure 1) that nodes of the network use to store unconfirmed transactions that have been broadcast to the network by applications. Applications send their transactions to one or more nodes of the network and nodes in turn disseminate the transactions across the network use gossip protocol. Network heterogeneities generally result in the transactions making it to the individual mempools in different orders. Transactions reside on mempools until they are batched up

for decentralized verification. PoRT protocol provides Network Discovery Service APIs that lets any entity (node, application) to query the status of mempools across the network, to track transaction propagation across mempools and monitor transaction handling latency.

The standardized schemas for transactions implicitly introduces entropy in every transaction that is sent to the network. With fields such as *timestamp*, *nonce*, *transaction ID* being unique and non-repetitive for a given application, a set of transactions sent by an application act as stores of entropy that the network can tap into to derive randomness in the system. The network dissuades duplicate transactions from being sent by levying penalties on the producers for such actions. The network checks for duplicate transactions being sent to the network by periodically comparing the transaction hashes with that of prior transactions. Producers who are repeatedly generating non-compliant transactions face rate limits and face the risk of eventually getting phased out.

Our solution uses batches of transactions as a source of entropy. While at the outset the above design may appear to invite *grinding attacks*, a two-way cryptographic sortition algorithm (explained subsequently) that PoRT incorporates will render such attacks unrewarding.

3.6 Parallel chains and Leader Election Function

PoRT protocol assigns all validated transactions sent to the network by an application to a single chain that is solely dedicated to that application. This design helps better management of application data ranging from faster responses to data query, efficient audits on transaction handling, until the eventual chain retirement when the application exits the platform. The network shall comprise as many independent chains as there are applications onboarded and hence the name, parallel chains. When an application a_i (with DID: id_i) is first onboarded onto the platform, the following steps are adopted:

- A dedicated chain whose DID is $\mathrm{H}(id_i)$ is created across all nodes of the network.
- The application specifies the size of storage required (in GB), region(s) of storage, billing frequency (monthly, bi-monthly, semi-annual, annual) and a replication factor (rf) for its transactions namely, $\{3, 5, 7\}$.

A leader is elected for the newly created chain and is deemed as the only entity who is allowed to propose blocks until a certain *inter-epoch block height* is reached for that chain, after which, a new leader is elected and process continues. An *epoch* is the time duration between two leader elections and *inter-epoch block height* is the number of blocks added to chain during that epoch. Both the leader election and the subsequent block height determination use Verifiable Random Functions. The above design prevents forks and facilitates instant finality, thereby helping the platform's throughput.

Whenever a chain c_x comes up for election, whose last block was b_x a node N_i with DID id_i generates a 256 bit random number r_i as r_i =H(DSA_Sign(H(b_x), sk_i)) which is mapped to number between $(0, 1)$, as $\hat{r_i}$ = P(r_i) (from (14)). A score w_i is computed as the weighted combination of the node's stake s_i and $\hat{r_i}$ in the form $w_i = a * s_i + b * \hat{r_i}$, where a and b are parameters pre-identified by the network, $0 < a < 1$, $0 < b < 1$ and $a + b = 1$. The inter-epoch block height is determined as $bh_i = Q(r_i, 1, max\_blocks)$ (from 15), where max_blocks is the network wide parameter that indicates the maximum number of blocks a leader can propose during one epoch on a chain.

The 256 bit random number r_i is a number that only N_i can generate and is not decipherable to the rest of the network until explicitly shared by the node. N_i can prove to the rest of the network that r_i was generated from $\mathrm{H}(c_x)$ by submitting proof π_i, where π_i = DSA_Sign(H($id_i \oplus \mathrm{H}(c_x)$), sk_i), that can be verified by the network upon unsigning and verifying that the message signed was derived from $\mathrm{H}(c_x)$. N_i disseminates its scores to the network (as illustrated in Table 2 through gossip sublayer.

The node that generated the highest w_i will be deemed as the leader for chain c_x, until bh_i new

Table 2. Leader election function (LEF) reults submitted by Node N_i

LEF Header	
Signature	Signed hash of [Header | Result Body]
Chain ID	H(id_i): DID of application a_i's chain
From address	DID URI of Node N_x
LEF Body	
Random number generated	r_i
Proof	DSA_Sign(H($id_i \oplus$ H(c_i))
Inter-epoch block height	bh_i
Stake held	s_i
Cumulative score	w_i

blocks were added to the chain. The index z of the leader is chosen as

$$z = \arg \max_{i \in (1..N)} a * s_i + b * \hat{r}_i \tag{1}$$

But the challenge is that in a partially synchronous network, there is a possibility that some nodes experience longer delays or even fail completely and as a result identifying the absolute maximum score as detailed in the equation above may be infeasible. PoRT protocol adopts a *time-out* period that is long enough to accommodate unforeseen network delays, but short enough to keep the protocol from not stalling, after the passage of which, nodes that deem that they have the highest overall scores declare themselves as the leader. If such a declaration is contested, then the leader is elected with a second round of consensus gathering.

The above design is verifiable. It gives nodes that held higher stakes a better chance of winning the leader election for the given chain, but also introduces uncertainty into leader election by introducing the weighted random number $\hat{r}_i$ in the equation. At a high level, the leader N_z serves the following roles for chain c_x until bh_z blocks are added to chain, namely:

- Aggregating transactions sent by application a_x from its mempool and creating batches of transactions that will be dispatched for verification
- Identifying subnets using *two-way cryptographic sortition* for (i) consensus gathering on the set of valid transactions (ii) subnet driven state machine replication for on-chain storage and CRUD friendly local data stores.

3.7 Two-way Cryptographic Sortition

Cryptographic sortition is a technique used in distributed systems to randomly select one or more participants from a group of potential candidates in a way that is both unbiased and verifiable. Consider the case when a leader node uses a source of randomness from within the network, and generates a random number that only they have the capacity to generate and repurposes the same to identify a subnet to execute one or more tasks. While it can be proven that the subnet was indeed created from the given random number, there isn't a way to rule out the scenario when the leader employed a grinding attack in identifying the optimal random number that identifies their fellow Byzantine nodes for the subnet.

PoRT protocol requires the individual nodes identified to serve the task(s) to run cryptographic sortition themselves, by generating their own random number that was derived from the *same source of randomness* as that of the leader, and determining for themselves if they are allowed to serve the assigned role or otherwise. In this design, the task assigner's grinding attack could be

rendered ineffective as they cannot predict the task receiver's allowed role to serve. The application of cryptographic sortition both by the task assigner and the task receiver is what we call as the *two-way cryptographic sortition*.

3.8 Batch creation, batch assembly and batch verification

Node N_z aggregates the unverified data transactions that were sent by application a_x (whose chain in c_x), from its mempool and creates a batch $B = \{dt_1, dt_2, ...dt_m\}$ comprising an ordered set of m transactions. Batch creation is governed by the most impinging of the three criteria namely: (a) maximum number of data transactions (b) maximum size of the batch (c) maximum wait-time allowed for data transactions. Using B as a source of entropy, N_z generates a random number r_z =H(DSA_Sign(H(B), sk_z)) and the VRF proof π_z =DSA_Sign(H(B), sk_z). Next, it identifies indices of p nodes who shall be the recipients of batch B by invoking Q(r_z, p, n) (from 15), where n corresponds to the number of nodes in the network. The leader shares (r_z, π_z) with the rest of the network to supply evidence that the indices in p were obtained from r_z.

Since all transactions dispatched to the network are propagated to mempools across the network, PoRT protocol keeps communication overheads associated with batch dispatch light by only requiring the leader to dispatch an ordered set of transaction IDs that comprise the batch and not the entire batch. The recipient nodes assemble the batch by retrieving the corresponding transactions from their respective mempools. If due to network delays, the recipient node finds one or more transactions missing from its mempool, they shall retrieve the missing transactions from their peers using DHTs for resource lookup.

Batch verification primarily entails running data transaction verification detailed in section 3.4.1 on the ordered set of transactions in a batch. It is important to establish that a node which, as per protocol, was supposed to run batch verification on a given batch B, performed the job independently and thoroughly without gathering the verification results through out-of-band communications with other fellow nodes who were tasked at verifying the same batch. If such maliciousness was left unchecked, then the Byzantine nodes may gain unfair computational advantage over honest nodes. PoRT protocol adopts a game theoretic construct named *batch derivatives* to such *freeloader attacks* in batch verification.

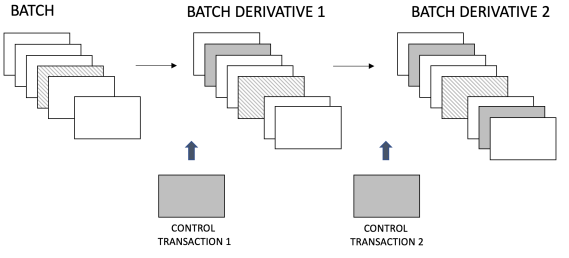

Figure 2. Control transactions that resemble data transactions sent by applications but are designed to fail data transaction verification are inserted at random indices. The original batch had contained one transaction that would not have passed verification.

3.8.1 Batch Derivatives

A *batch derivative* B' is obtained by inserting one or more non-conforming transactions called the *control transactions* at random indices onto B. A control transaction is similar to a regular transaction in structure, but is designed to fail verification either due to source authentication problems or due to data integrity issues. The control transactions are dispatched to the network by the network nodes and hence they make it to the mempools across the network just like regular transactions do. Let

$C = \{c_1, c_2, ...c_\gamma\}$ be the set of γ control transactions. $B' = B \odot C$ is a batch derivative constructed upon inserting them at random locations onto B, such that the data transactions in B still appear in the same relative order. The operator $\odot$ signifies control transaction ingestion. Figure 2 illustrates the creation of B'.

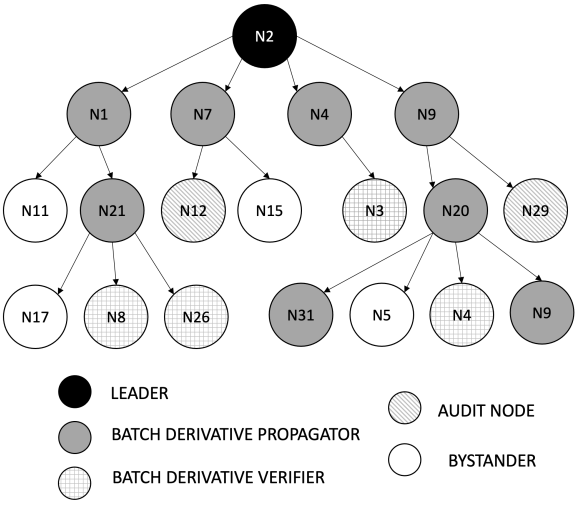

Figure 3. An illustration of the dynamic subnets for batch verification

3.8.2 Two-way Sortition driven Dynamic Subnets

Batch verification is a multi-stage process, the evolution dynamics of which is unpredictable. Each node that received a batch (or a batch derivative) can perform any one of 4 roles in batch verification namely, (a) Verify the received batch and provide verification results to the leader (through direct port-based communication) and the rest of the network (through gossip) (b) Create a batch derivative and propagate the same (c) Perform behavioral audits, monitoring the VRF proofs submitted and the overall adherence to protocol (d) Assume the role of a by-stander, with no real role to play in verifying the batch.

Stage #1 of batch verification proceeds as follows: Every one of the p nodes that leader N_z had identified to take part in verifying batch B, perform the following tasks:

- Step #1: Each node takes (r_z, p_z) submitted by N_z and verifies N_z's source of randomness and if their node index was indeed derived from r_z as prescribed by the protocol. If incongruencies are detected, then the node reports the non-conformity observed.
- Step #2: Nodes in this stage of batch verification assume the role of batch derivative creation and propagation. Each node uses batch B as a source of entropy and generates its own random number $r\prime$ (signing with its own private key) and the associated proof of randomness. It invokes $Q(r\prime, 1, p)$ and determines the the number of nodes q that is should dispatch batch derivatives to. Next, it invokes $Q(r\prime, q, n)$, identifies the q recipient node indices, creates different batch derivatives $\{B\prime_1, B\prime_2, ..., B\prime_q\}$ for each recipient by inserting a control message at a random index onto B and dispatches them their respective batch derivatives.

Stage #2 of batch verification proceeds as follows:

- The recipient nodes in the second stage of batch verification, again establish protocol conformity as detailed in Step #1 above.

- Nodes decipher their respective roles based on the random number rl that they generate using the same entropy source as the leader did, B. Each node invokes $\mathrm{P}(rl)$, which maps rl to a probability score $\hat{s}$. Based on $\hat{s}$, the node identifies the role it is allowed to serve:

$$\begin{aligned} 0 \leq \hat{s} < 0.2 &\implies \text{dispatch batch derivatives} \\ 0.2 \leq \hat{s} < 0.6 &\implies \text{verify batch derivative} \\ 0.6 \leq \hat{s} \leq 0.8 &\implies \text{serve as a bystander} \\ 0.8 \leq \hat{s} < 1 &\implies \text{run behavioral audits} \end{aligned}$$

- Batch derivative dispatch is detailed in Step #2 above.
- As a batch derivative verifier, nodes run the data transaction verification function on every transaction in the batch derivative and submit their results (Table 3 to the network). Nodes deploy compute sharding, thereby identifying segments of the batch that they will run batch verifications for. These segment indices are also determined from the random number that they generated.
- As a bystander, the node serves no role on the received batch derivative.
- As an audit node, the node corroborates source of randomness, VRF proofs and overall adherence to protocol.

Figure 3 illustrates dynamic subnet creation for batch verification. Batch verification runs for one

Table 3. Summary from Batch Derivative Verifier

Result Header	
Signature	Signed hash of [Header | Result Body]
Batch ID	Hash(B): Hash of the original message
From address	DID URI of Node N_j
Result Body	
Role	Batch Verifier
Proof π_x	$< h(B), PvK_x >$
Assigner	DID URI of node N_i that assigned the batch derivative
Batch derivative	B' : Batch derivative that was verified
Segments reviewed	1, 3, 4
Verification result	$h(M'_1), h(M'_3), h(M'_4)$
Indices of failed messages	Indices where message verification failed

additional (and final) stage. The leader N_z gathers verification results submitted and gathers the list of verified messages in the batch and creates a *block* of verified transactions that shall be added to chain c_x. The above design renders adaptive attacks ineffective and disincentivizes collusion.

3.9 Decentralized Storage: LogChains and Local Data Stores

A block $\hat{B}$ comprises an ordered set of verified data transactions. The header of the block comprises a Merkle Root, derived from a hash tree (Merkle Tree) of all the ordered data transactions contained in the block. All the transactions in $\hat{B}$ originated from application a_x, are destined to be stored on the application-specific chain c_x, and will have to be replicated a minimum of rf_x times to meet the application's requirements. PoRT's solution towards decentralized storage encompasses the following:

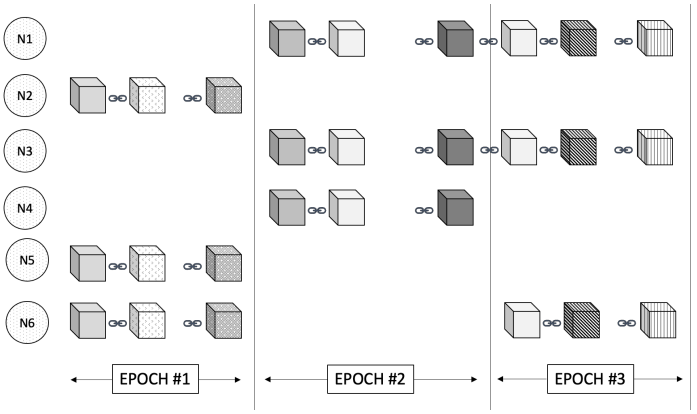

Figure 4. An illustration of blocks stored in epoch based static storage subnets

3.9.1 State Machine Replication using Subnets

State Machine Replication is executed in a subnet context where two subnets are created, one for chain-based storage (called Log-Chain store (Figure 4) and the other for a CRUD friendly storage (called Data store). The subnets are created using two-way cryptographic sortition with full verifiability. The size of the Log-Chain store subnet is $f + 1$ (where in a n node network we have assumed that the number of Byzantines f are such that n 3f + 1). The size of the data store subnet is the same as the replication factor rf_x chosen by the application. The subnet's term period is the same as that of the leader for the chain. After bh_z blocks (called inter-epoch block height) have been added to chain, both the subnets are retired. These are called *epoch based static subnets* for decentralized storage, ones that are commissioned into existence at the beginning of a leader's term and decommissioned from chain storage responsibilities at the end of the leader's term.

3.9.2 Epoch based Static Subnets for Storage

The entropy contained in the first block $\hat{B}$ proposed by the leader for the chain during the epoch is as the source of entropy. N_z generates a random number $\hat{r}_z$ =H(DSA_Sign(H($\hat{B}$), sk_z)) and the VRF proof $\hat{\pi}_z$ =DSA_Sign(H($\hat{B}$), sk_z). The leader executes the first step in the two-way cryptographic sortition step. Next, it identifies the indices of $f + 1 + rf_x$ nodes by invoking Q($\hat{r}_z$, $f + 1 + rf_x$, n), the first f+1 nodes getting invited to serve Log-Chain storage for chain c_x and the remaining rf_x to serve as the local data store for the same chain for the epoch. The leader supplies $\hat{r}_z$ and $\hat{\pi}_z$ along with the respective invitations.

The recipient nodes establish protocol conformity (as detailed in Step #1 in the previous subsection) whereby they establish that there was no malicious behavior or protocol non-conformity in them getting the invitations to join the epoch based static subnet. The nodes apply cryptographic sortition themselves, generate their own random number $\hat{r}\prime_z$ (derived from $\hat{B}$). If P($\hat{r}\prime_z$) $\geq$ 0.5, then they accept the invitation. If P($\hat{r}\prime_z$) < 0.5, then the node invokes Q($\hat{r}\prime_z$, 2, n) and identifies new recipient nodes for the invitation. The nodes submit their random numbers and the corresponding proofs to the leader and to the rest of the network.

Once the subnets have been formed, every block dispatched by the leader gets replicated within the LogChain subnet on the respective chains dedicated for application a_x. As a measure of *safety*, a Merkle root is derived, whereby the hash of the blocks added to chain in the current epoch form the leaf nodes of the Merkle tree, every time a new block is added to chain and stored on the respective block headers. The transactions added to the block are simultaneously maintained in local data store preserving the order of the transactions. A data transaction of type *create* results in a new record added

to the local data store. Transactions of type *update* result in records getting updated as requested. Transactions of type *read* cause no change to the state of the data store and those of type *delete* get expunged from the data store. Apart from the two forms of storage detailed above, a third network wide chain that leader N_z presides for the epoch duration is maintained that logs the hash of the chain at the end of every epoch, yeilding network-wide *checkpointing* capabilities.

3.10 Safety, Liveness and Byzantine Defense Mechanisms

Table 4 discusses the different Byzantine attack vectors that PoRT protocol defends against. The carefully laid out design choices starting from standardized message schema, network discovery service APIs, batch derivatives, two-way cryptographic sortition and dynamic subnets contribute towards keeping adaptive attacks generally ineffective. The *safety* guarantees of the protocol stem from the *no fork* parallel chain construct that has a leader elected for an epoch duration. Further, dynamic audits and protocol governed checkpointing in the form of Merkle roots derived for every newly added block to the respective chains further strengthen the safety claims of PoRT protocol.

For *liveness* guarantees, PoRT protocol adopts the following measures: if a node that was inducted into either the dynamic subnets for batch verification or the static subnets for decentralized storage crashes or simply becomes unresponsive, to keep the protocol moving, the node is replaced by its nearest node (applying XOR distance on the hash of the node's DID) in the Kademlia keyspace. The same applies to the leader. If the leader goes down or becomes unresponsive or displays malicious behavior, nodes agree to initiate a lighter version of the *view change* protocol. As against the three phases of the protocol (preparation phase, proposal phase and acceptance phase), just the preparation phase is initiated and nodes need to come to a consensus that the primary leader is non-reachable. Then, in deterministic fashion, the node that was closest to the leader node in the keyspace is appointed to take up the leader's responsibilities.

There could be instances when the two-way sortition driven static subnet creation process could run into far more rounds than usual, either due to the distribution of random numbers generated by the potential candidates or due to unfavorable network conditions. PoRT employs a *time-out* duration after which the candidates for unfilled spots in the subnet are identified using deterministic methods such as keyspace proximity to leader. These measures are integrated to preserve *liveness*.

4. Discussions and Future work

Kandola, powered by PoRT protocol, empowers applications to define their own data schema, secures data at source, preserves data integrity and privacy all through the data life-cycle and wrests data ownership and control at the hands of the rightful owners. As a decentralized data exchange platform built for serving real-time data needs of applications, Kandola shall unlock the many potentials of decentralized data marketplaces. Users can buy and sell data for the development of data-driven services. Garnering massive datasets for training AI models with the right permissions from the respectie data owners can further the already interesting AI landscape.

Beyond facilitating data pipeline markets, Kandola's approach encourages the development of open standards. IoT device manufacturers, for example, could agree on data standards, ensuring that devices across different manufacturers adhere to a unified schema. This standardization allows solution developers to build applications based on the schema without being restricted to a specific device or manufacturer, ultimately fostering a larger marketplace for both parties (e.g., the Tuya IoT Ecosystem). Kandola's infrastructure layer has the potential to make a substantial positive impact across various industries and domains. By creating a fair marketplace and offering strong economic incentives for specialized storage providers, the platform unlocks exciting possibilities for the future.

Our Team is currently building a TestNet platform that will be run by PoRT consensus. We will run extensive tests with real-life data streams being sent to the platform and will share our performance benchmarks with the community.

Table 4. PoRT protocol's defense against attack vectors

Attack vector	PoRT Defense
Replay attack: Producer burdens the network with duplicate, messages undermining the source of entropy and stressing the network	Nodes run hash-collision test on periodically, on incoming messages by comparing their hash with that of mempool messages
Sybil attacks: A malicious node operates under multiple identities simultaneously	In a stake based network with role fluidity, high collateral needed for Sybil attack is less rewarding
Censorship attack: The leader for the chain withholds messages from certain producers, not including them in a batch	Network Discovery Service API's let any node run queries off of leader's mempool and detect instances of withheld messages.
Batch incongruence: The leader or batch derivative propagators may add or remove or shuffle native messages and delay consensus	Audit nodes check for the hashes of batch derivatives by expunging control messages and trace origins of batch incongruence
Freeloadership: Fellow Byzantine nodes may establish out-of-band communication and share answer keys	Dynamic subnets introduce role unpredictability rendering such collusion less viable or effective
Grinding attacks: A leader could manipulate blocks such that their own re-election or that of their fellow and the Byzantine is favored	Two-way sortition in conjunction with batch derivatives makes network processes less predictable and hence more secure
Adaptive attacks: Malicious nodes may stage coordinated attacks on certain nodes affecting their ability to serve a role and delaying consensus	Dynamic subnets diminish success of behind coordinated attacks as roles are non-decipherable to the rest of the network
Non-conformity in roles: Malicious nodes may perform operations that are beyond their scope	Audit nodes periodically verify role conformity. Non-conforming nodes could loose stake.

References

0xphillian and Fundamental Labs. 2022. *Decentralized Storage: A Pillar of Web3.* https://6pjecoitbb3mbacc67rvmct3gbugplnty5ptok4t6tkgvxnoj2ya.arweave.net/89JBORMIdsCAQvfjVgp7MGhnrbPHXzcrk_TUat2uTrA.

Al-Bassam, Mustafa, Alberto Sonnino, Shehar Bano, Dave Hrycyszyn, and George Danezis. 2017. Chainspace: A Sharded Smart Contracts Platform. *ArXiv* abs/1708.03778.

Benet, Juan. 2014. IPFS - Content Addressed, Versioned, P2P File System. *ArXiv* abs/1407.3561.

Berger, Christian, Signe Schwarz-Rüsch, Arne Vogel, Kai Bleeke, Leander Jehl, Hans P. Reiser, and Rüdiger Kapitza. 2023. SoK: Scalability Techniques for BFT Consensus. *ArXiv* abs/2303.11045.

Camenisch, Jan, Manu Drijvers, and Timo Hanke. 2022. Internet Computer Consensus. *Proceedings of the 2022 ACM Symposium on Principles of Distributed Computing.*

Crust. 2020. *Crust: White Paper v1.9.9.* https://crust-data.oss-cn-shanghai.aliyuncs.com/crust-home/whitepapers/whitepaper_en.pdf.

Gilad, Yossi, Rotem Hemo, Silvio Micali, Georgios Vlachos, and Nickolai Zeldovich. 2017. Algorand: Scaling Byzantine Agreements for Cryptocurrencies. *Proceedings of the 26th Symposium on Operating Systems Principles.*

Guo, Bingyong, Yuan Lu, Zhenliang Lu, Qiang Tang, Jing Xu, and Zhenfeng Zhang. 2022. Speeding Dumbo: Pushing Asynchronous BFT Closer to Practice. *IACR Cryptol. ePrint Arch.* 2022:27.

Guo, Bingyong, Zhenliang Lu, Qiang Tang, Jing Xu, and Zhenfeng Zhang. 2020. Dumbo: Faster Asynchronous BFT Protocols. *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security.*

Hassanzadeh-Nazarabadi, Yahya, Sanaz Taheri Boshrooyeh, Safa Otoum, Seyhan Uçar, and Öznur Özkasap. 2021. DHT-based Communications Survey: Architectures and Use Cases. *ArXiv* abs/2109.10787.

Kokoris-Kogias, Eleftherios, Philipp Jovanovic, Linus Gasser, Nicolas Gailly, Ewa Syta, and Bryan Ford. 2018. OmniLedger: A Secure, Scale-Out, Decentralized Ledger via Sharding. *2018 IEEE Symposium on Security and Privacy (SP),* 583–598.

Labs, Protocol. 2017. *Filecoin: A Decentralized Storage Network.* https://filecoin.io/filecoin.pdf.

Labs, Storj. 2018. *Storj: A Decentralized Cloud Storage Network Framework.* https://www.storj.io/storjv3.pdf.

Luu, Loi, Viswesh Narayanan, Chaodong Zheng, Kunal Baweja, Seth Gilbert, and P. Saxena. 2016. A Secure Sharding Protocol For Open Blockchains. *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security.*

Mao, Yifan, Soubhik Deb, Shaileshh Bojja Venkatakrishnan, Sreeram Kannan, and Kannan Srinivasan. 2020. Perigee: Efficient Peer-to-Peer Network Design for Blockchains. *Proceedings of the 39th Symposium on Principles of Distributed Computing.*

Maymounkov, Petar, and David Mazières. 2002. Kademlia: A Peer-to-Peer Information System Based on the XOR Metric. In *International workshop on peer-to-peer systems.*

Miller, Andrew K., Yuchong Xia, Kyle Croman, Elaine Shi, and Dawn Xiaodong Song. 2016. The Honey Badger of BFT Protocols. *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security.*

Neiheiser, Ray, Miguel Matos, and Luís E. T. Rodrigues. 2021. Kauri: Scalable BFT Consensus with Pipelined Tree-Based Dissemination and Aggregation. *Proceedings of the ACM SIGOPS 28th Symposium on Operating Systems Principles.*

Rana, Ranvir, Sreeram Kannan, DavidN C. Tse, and Pramod Viswanath. 2022. Free2Shard. *Proceedings of the ACM on Measurement and Analysis of Computing Systems* 6:1–38.

Rocket, T. J., Maofan Yin, Kevin Sekniqi, Robbert van Renesse, and Emin Gün Sirer. 2019. Scalable and Probabilistic Leaderless BFT Consensus through Metastability. *ArXiv* abs/1906.08936.

Stathakopoulou, Chrysoula, Tudor David, and Marko Vukolic. 2019. Mir-BFT: High-Throughput BFT for Blockchains. *ArXiv* abs/1906.05552.

Swarm. 2021. *Storage and Communication Infrastructure for a Self-Sovereign Digital Society.* https://www.ethswarm.org/swarm-whitepaper.pdf.

Vorick, David. 2014. Simple Decentralized Storage.

Williams, Samuel, and William Jones. 2018. *Arweave Lightpaper Version 0.9.* https://whitepaper.io/document/627/arweave-whitepaper.

Yin, Maofan, Dahlia Malkhi, Michael K. Reiter, Guy Golan-Gueta, and Ittai Abraham. 2019. HotStuff: BFT Consensus with Linearity and Responsiveness. *Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing.*

Zamani, Mahdi, Mahnush Movahedi, and Mariana Raykova. 2018. RapidChain: Scaling Blockchain via Full Sharding. *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security.*