Timbre: A Decentralized Forum with Blockchain-based Distributed Data Store

Guyu Fan Xin Tong Raluca-Georgia Diugan Jay Chen

{gf940, xt405, rgd248, jc2977}@nyu.edu

Abstract

Much of the public discourse today takes place online, especially on social media platforms. However, these platforms are almost always subject to censorship. For instance, posts on Facebook and tweets on Twitter may be removed if they are deemed inappropriate by the service providers, and threads and comments on Reddit may be locked or deleted at will by subreddit moderators. Centralized storage of user content makes it convenient to monopolize moderation power, and the fact that the average user only has a partial view of all content exacerbates the problem. In response to these issues, we design and implement Timbre, a Blockchain-based distributed data storage system for decentralized online forums. In Timbre, forum content is distributed to different nodes, and a blockchain that records the complete history of user data storage and retrieval transactions is disseminated across the peer-to-peer (P2P) network. We also introduce the concept of blockchain epochs to prevent concentration of power in the system. Ultimately, we aim to provide an open, trustworthy and censorship-resistant system for public online discourse.

Introduction

A **Timbre** forum is a decentralized online discussion platform where data storage is distributed among participating nodes. A Timbre blockchain is the backbone of the forum. It keeps track of forum posts and distributes rewards to nodes that contribute to the forum. New forum posts are collected by miners and distributed to storage providers. Information about forum post storage is recorded in the blockchain. As the forum grows, its blockchain is extended.

Paper Outline

In the **Timbre Design** section, we give an extensive description of Timbre's blockchain, content management and economics design. In the **Timbre Protocols** section, we detail the protocols for making posts, matching posts with storage providers, creating storage deals, verifying storage deals and disseminating the blockchain. In the **Timbre Analysis** section, we present an analysis of our system design by answering questions about significant design decisions on the blockchain structure, content storage and reward mechanisms. Finally, we conclude with **Future Work** and discuss improvements that can be made to our system.

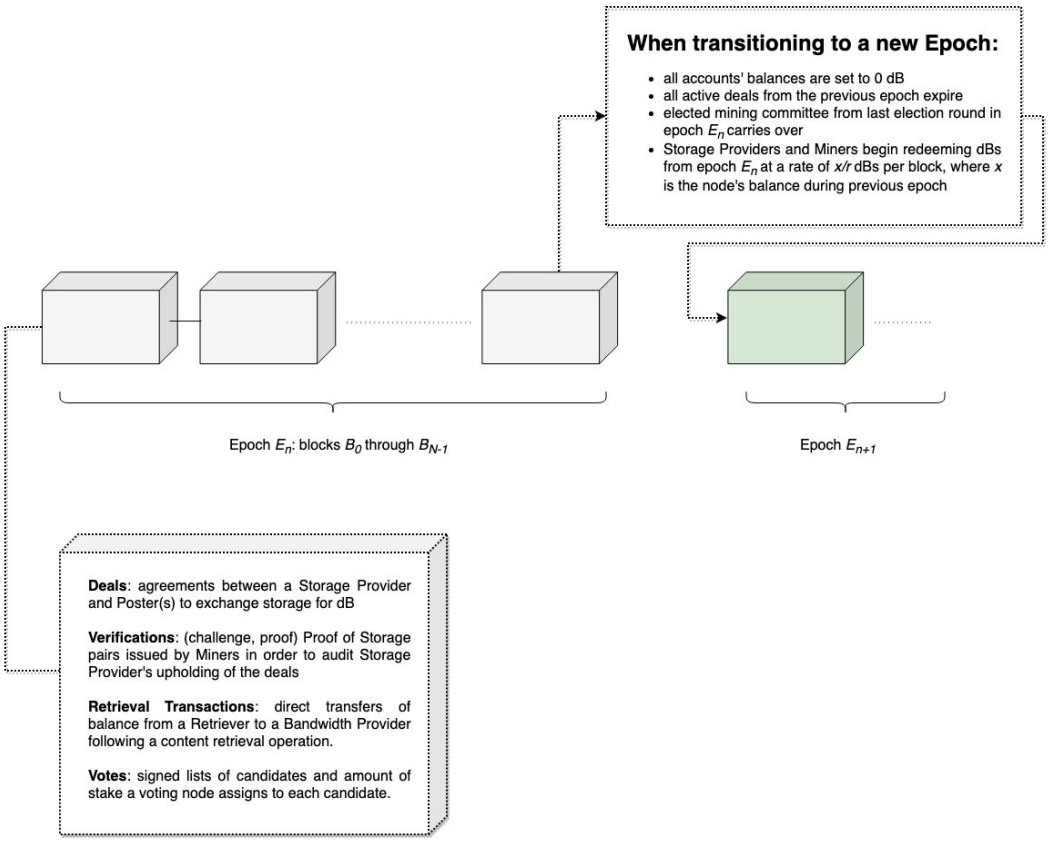

Figure 1: Blockchain Design.

Timbre Design

In this section, we describe the overall design of Timbre without delving into too many implementation details.

Roles

Nodes in the Timbre P2P network may take on one or more distinct roles. *Posters* submit new posts to the forum, and these posts are stored by *storage providers*. *Retrievers* download posts from *bandwidth providers* that serve the posts from their local cache. Finally, *miners* are responsible for coordinating certain operations and recording the results in order to extend the Timbre blockchain.

In the centralized setting, an average user of a forum would only be a poster and a retriever. However, to ensure that Timbre functions smoothly, it is crucial that most users also become storage providers and bandwidth providers, and a small number of users become miners. Therefore, one of our central design goals is to build a set of protocols that incentivize forum users to fulfill these necessary responsibilities.

Blockchain

Every Timbre forum is implemented on top of a blockchain (see figure 1), which acts as a public ledger that keeps track of forum posts. The cryptocurrency (*decibles*) associated

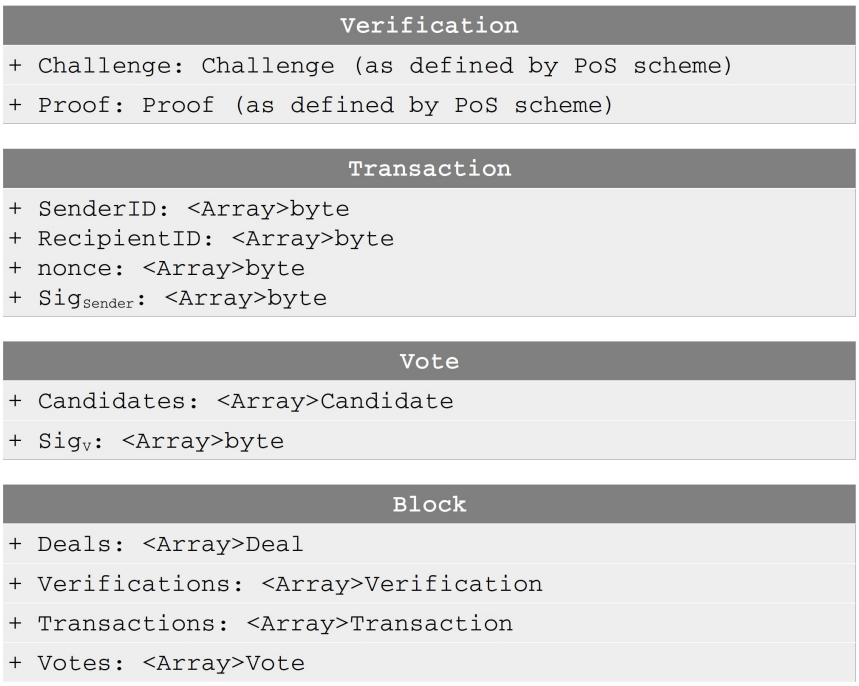

Figure 2: Block Structure.

with the blockchain rewards storage providers, bandwidth providers and miners for their contributions to the network.

Block Structure

Each block is divided into four sections: storage *deals*, deal *verifications*, retrieval *transactions*, and *votes*. Conceptually, a deal is an agreement between a storage provider and one or more posters to exchange post storage for decibles. Deals are not permanent, and a deal is *active* before it expires. Miners are responsible for sampling active deals on the blockchain and checking if the storage providers are correctly storing the posts in those deals. A miner generates a verification using a Proof of Storage scheme [1] for each sampled active deal. The storage provider's income depends on the success of the verification. A retrieval transaction, similar to transactions in other cryptocurrency systems, is a direct transfer of balance from one account to another. A retriever pays a bandwidth provider by sending a retrieval transaction to the miners. Finally, nodes send votes to the current mining committee to elect the next mining committee (mining committee is defined in section 3.2.2). The block structure can be viewed in Figure 2.

Consensus

Most existing blockchain consensus algorithms operate under the assumption that honest nodes own the majority of some resource in the network. For proof-of-work consensus, that resource is hashing power; for proof-of-stake consensus it is token ownership. In Timbre, we use Delegated Proof of Stake (DPoS), a variant of proof-of-stake consensus, because DPoS typically has higher transaction throughput, achieves a greater degree of centralization and is not as energy-intensive as proof-of-stake consensus [2].

In DPoS, an elected *mining committee* of $\leq K_{max}$ (system parameter) miners is responsible for producing blocks. Any node can declare its candidacy for the mining com-

mittee. Any node with a positive decibel balance can vote by splitting its stake among at least K_{max} candidates. The candidates that receive the top K_{max} stake *and* at least a fixed fraction ($1/s$, system parameter) of the total stake in the network become the mining committee. At the beginning of a new Timbre forum there may be $< K_{max}$ nodes, therefore the committee size could also be $< K_{max}$. As more nodes join the forum and gain stake, the committee size will dynamically increase when a new node receives $\geq 1/s$ stake. In the case where no nodes receive $\geq 1/s$ stake, K_{min} (system parameter s.t. $1 < K_{min} < K_{max}$) number of nodes with the highest stakes get to be the next mining committee.

Once a committee is elected, a mining *round* starts. A mining round is divided into *slots*. Each slot lasts T_s (system parameter) seconds, and each elected miner is assigned exactly one slot to produce a new block. A miner's block is skipped if the miner fails to produce the block within its assigned slot. After the mining round ends, votes are tallied, a new committee is elected and the next mining round begins.

Epochs

Scalability and concentration of power are issues common to many blockchain systems. In Timbre, the solution is to divide the blockchain into *epochs*. Each epoch contains a fixed number (N, system parameter) of blocks. Every node's decibel balance is reset to zero after the start of a new epoch, but epoch transitions do not affect mining committee elections. In other words, miners with the most votes in the last round of the previous epoch still become the first mining committee of the new epoch.

Since all storage deals expire at the end of the epoch during which they were made, towards the end of every epoch system throughput in terms of posting is expected to drop. Then at the beginning of the next epoch, posting rate will gradually go back to normal as nodes earn new tokens.

A node may redeem a fixed fraction ($1/r$, system parameter) of its decibels from the previous epoch at a fixed rate by being a miner or a storage provider in the new epoch. Suppose that a node had x decibels in the previous epoch. Then at each block in the new epoch, the node may redeem x/rN decibels if the node is the current miner or the storage provider for an active deal.

Network

A P2P network provides two basic functionalities required by every Timbre forum: *broadcasting* and *one-to-one messaging*. A node may, depending on the use case, broadcast a message globally or only locally to other nearby nodes. For instance, a miner should broadcast a new block globally, while a storage provider should only advertise its price locally to avoid congesting the network. A node may also send a one-to-one message to some other node in the network. For example, a poster should send its `PostRequest` directly to the current mining committee.

Timbre uses a Distributed Hash Table (DHT) for both communication patterns. The DHT-based broadcast protocol proposed by [3] is well suited for both global and local broadcasts, and the DHT itself serves as a natural routing table for one-to-one messaging. To prevent spamming in the network, nodes should implement rate limiting and blacklist mechanisms.

Some operations in the network is time-sensitive. For instance, the mining committee

needs to agree on the exact start and end times of each mining slot. The Network Time Protocol (NTP) may be used to synchronize time between different nodes.

Only K_{min} number of nodes are needed to bootstrap a Timbre forum. The bootstrapping nodes can announce the forum to others in whichever way they like, but in particular they can submit their addresses and forum information to the central directory of publicly known forums that we Timbre developers maintain. The bootstrappers can later publish more known addresses in their network, and new nodes can join by contacting the set of available addresses.

Logically speaking, every Timbre forum maintains its own network independent of other forums, but in practice, a single node may participate in multiple forums, resulting in a sparsely connected mega-network of all Timbre nodes.

Content Management

The contents that a Timbre forum contains are user-generated posts. In this subsection, we introduce post creation, aggregation, storage and retrieval procedures in a Timbre network.

Posts

There are two kinds of posts in Timbre: *root* posts and *reply* posts. A root post is not a reply to any existing posts in the forum and creates a new *thread*, while a reply post is a reply to some existing post and therefore belongs to an existing thread. Root posts and reply posts are handled differently in Timbre, but they both go through the same lifecycle.

First, a poster broadcasts its post in its neighborhood and sends a `PostRequest` to the mining committee. A miner then forwards the `PostRequest` to an appropriate storage provider. After receiving the storage provider's confirmation, the miner records a `Deal` containing the `PostRequest` in the new block. While the deal is active, miners periodically check if the storage provider is faithfully storing the post, and retrievers may reach out to the storage provider to download the post. After the deal expires, the storage provider is no longer obligated to store the post. However, there might still be nodes caching the post locally on their machines, so it is still possible to retrieve a post even though the deal has already expired.

Storage

Storage providers offer storage in *chunks*. Each chunk is M (system parameter) bytes and is dedicated to the storage of a particular thread. When a storage provider accepts a new root post, it allocates a chunk for the thread created by that root post. Miners will direct all future reply posts under that thread to the same storage provider until the chunk is filled. When the chunk is full, the next reply post will trigger another chunk allocation from either the same storage provider or some other storage provider.

At any given time, at most one storage provider is responsible for storing reply posts in a particular thread, and that storage provider is that thread's *primary* storage provider. Note that a thread may have *no* primary storage providers at all if the thread's posts perfect fill all the storage chunks. In this case, the storage provider that filled its chunk most recently is called the *previous primary* storage provider.

Each storage provider has a *total capacity*, which is the maximum amount of storage that it can provide. The collection of all chunks allocated by the storage provider is called its *committed storage*. The collection of all deals stored by the storage provider is called its *used storage*.

Each poster also stores its own posts. However, self-storage is not recorded on the blockchain as separate deals; it is only inferred from the poster identity field in the post request, and thus is not enforced by the system.

Retrieval

Bandwidth providers, like storage providers, also advertise their prices by local broadcasts. But while storage providers are compensated by storage deals, bandwidth providers earn decibels from retrieval transactions. Retrieval transactions are made on good faith and should be aggregated over time to keep the blockchain small. If a retriever fails to pay its due, the retrieval provider can simply blacklist that retriever and reject its future requests.

In Timbre, we assume that retrievers access forum content by threads. A retriever learns of existing threads in a Timbre forum by examining the storage deals in the blockchain. Once the retriever locates the deals of a particular thread, it may retrieve the thread by asking storage providers and posters or by locally broadcasting a retrieval request in the network.

A thread's most natural bandwidth providers are its storage providers. Storage providers also act as *tracker* nodes by maintaining a list of recent retrievers for that thread. If a storage provider has limited bandwidth or is not storing the thread because the deal has expired, then it can still redirect the retrieval request to several recent retrievers. If a storage provider is offline, then the retriever can either reach out to the individual posters in a thread, or broadcast its retrieval request locally to look for a nearby node with a cache of the thread.

Economics

In a Timbre forum, nodes accumulate and redistribute its `decibels` while carrying out the tasks of its roles. The blockchain records all valid transactions of the cryptocurrency, therefore each node can compute everyone's balance from the chain.

Reward Types

Upon mining a new block, the miner of the Timbre blockchain gets all but the last one of the following rewards by *system inflation*:

- A fixed *block reward* (b, system parameter) for creating the new block.
- *transaction fees*, a fixed percentage (t, system parameter) of the value of each storage deal and retrieval transaction it includes in its block.
- A fixed *verification reward* (v, system parameter) for including each *successful* deal verification in its block.
- A portion of its total amount to redeem from the last epoch (if any).

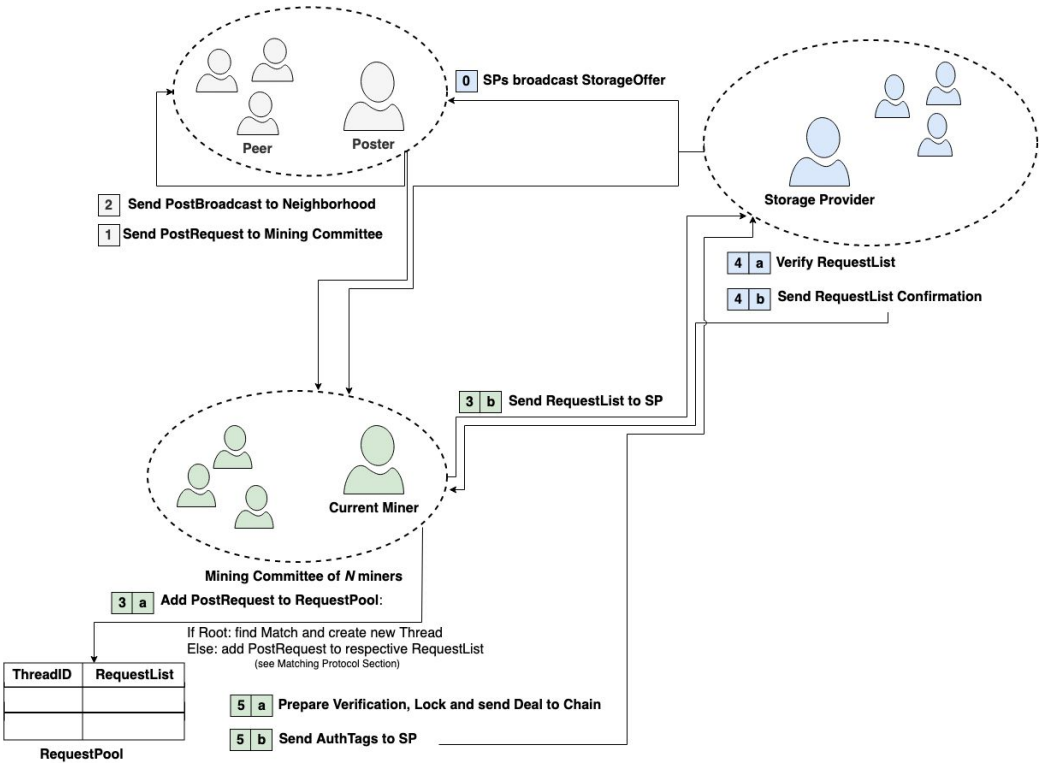

Figure 3: Summary of Posting, Matching and Verification Setup Protocol.

Intuitively, storage providers are paid from storage deals and bandwidth providers are paid from retrieval transactions. Posters and retrievers do not earn anything, so they have to also take on one or more of the rewarded roles to fully participate in the system.

Balance Calculation

Each Timbre node calculates and stores all node balances up to the latest block it knows. Notably, all payments related to a storage deal happen *block by block* until that deal expires; others happen instantly at the block they are included in. Therefore, once a block is verified, the included block rewards, verification rewards and retrieval transactions are deducted from the payers' balances (if not system inflation) and added to the receivers' *in full*. As for storage deals, the deal payment is also deducted in full from the poster's balance at once. Then at each following block, if the deal is verified successfully, the storage provider and the deal-making miner get their rewards for that block, and the storage provider gets partially redeemed; otherwise, no rewards are given, the storage provider does not get redeemed and the poster gets a refund for that block.

Protocol

This section describes various Timbre protocols in detail. Unless otherwise stated, prices are decibels *per unit spacetime*, duration is measured in *number of blocks* and timestamps represent date and time. A specification of certain protocol `Structure`s is shown in figure 4.

StorageOffer
+ StorageProviderID: <Array>byte + Cost: int + MaxDuration: int

PostRequest	
+ PosterID: <Array>byte	
+ PostHash: <Array>byte	
PostMetadata	+ PosterID: <Array>byte + PostSize + ReplyTo: PostHash + Timestamp: Daytime
+ PostContent: String	
+ MaxCost: int	
+ Duration: int (root post only)	
+ $Sig_{request}$: <Array>byte	

PostBroadcast	
+ PostContent: String	
PostMetadata	+ replyTo: PostHash + Timestamp: Daytime
+ $Sig_{broadcast}$: <Array>byte	

RequestList
+ Requests: <Array>PostRequest

RequestPool
+ RequestLists: <Array>RequestList

Deal
+ RequestList: RequestList (excluding PostContent in each PostRequest)
+ Name: String
+ PublicKey : <Array>byte
+ Sig_{SP}: <Array>byte

AuthTags
- Tags: <Array>byte

Figure 4: Protocol Structures.

Posting Protocol

The poster follows the posting protocol to make a new post.

1. To make a new root post, the poster must decide on `MaxCost` and `Duration`. `MaxCost` is the highest price that the poster is willing to pay to store the post. `Duration` is the poster's desired post storage duration. The poster can detect price fluctuations in the storage market by monitoring `StorageOffer`s broadcasted by storage providers and by examining recent `Deals` on the blockchain. This information will help the poster choose an appropriate `MaxCost`.

2. To make a new reply post, the poster does not specify `MaxCost` or `Duration` if there exists a primary storage provider for the reply post's thread. The primary storage provider is already storing some existing posts from the same thread. Therefore, the cost will be the same as the cost of the existing posts, and `Duration` will be adjusted to make the new reply post expire at the same time as the existing posts. In the rare case where there is no primary storage provider for the reply post's thread, the poster is free to adjust the `MaxCost` but should still omit the `Duration` to ensure that posts under the same thread stored across different chunks expire at the same time.

3. The poster generates `PostMetadata` which contains `PosterID`, `PostSize`, `Timestamp` and `ReplyTo`. `PostSize` is the size of `PostContent` and `PostHash`. If the new post is a reply post, `ReplyTo` is set its parent post's `PostHash`. `ReplyTo` is left empty for a new root post.

The next steps apply to both root posts and reply posts.

4. The poster hashes `PostContent` and `PostMetadata` to `PostHash`.

5. The poster generates $\texttt{Sig}_{\texttt{request}}$ by signing `MaxCost`, `Duration`, `PostMetaData` and `PostHash`.

6. The poster generates `PostRequest` containing `PosterID`, `PostHash`, `PostMetaData`, `PostContent`, `MaxCost`, `Duration` and $\texttt{Sig}_{\texttt{request}}$.

7. The poster sends the `PostRequest` to the mining committee.

8. The poster generates $\texttt{Sig}_{\texttt{broadcast}}$ by signing `PostContent` and `PostMetaData`.

9. The poster generates and locally broadcasts `PostBroadcast`.

Matching Protocol

A miner uses the matching protocol to match `PostRequest`s to storage providers. The miner's collection of `PostRequest` that have not been recorded on the blockchain yet is that miner's `RequestPool`.

1. The miner rejects a `PostRequest` if its `PostSize`, `PostHash` or $\texttt{Sig}_{\texttt{request}}$ is invalid.

2. The miner groups the `PostRequest`s in its `RequestPool` by threads. Each group is a `RequestList`.

3. Miners, like posters, also closely monitors the storage market. If a `RequestList` contains a single root `PostRequest`, then the miner chooses a compatible storage provider that minimizes the price difference between the `StorageOffer` and the `PostRequest`. A compatible storage provider is one with a compatible `StorageOffer`.

4. If a `RequestList` contains one or more reply `PostRequest`s under an existing thread, then the miner must choose that thread's primary storage provider if the primary storage provider exists. In the rare case where there is no primary storage provider, the miner is free to choose any compatible storage provider, but the miner should give preference to the previous primary storage provider while minimizing the price difference.

5. After choosing a storage provider, the miner knows the size of the remaining chunk allocated the the thread. If the `RequestList` exceeds the thread's remaining chunk size, the miner truncates the `RequestList` so that it fills the remaining chunk and discards the leftover `PostRequest`s, which should be handled by the next miner.

6. The miner sends the (possibly truncated) `RequestList` to the chosen storage provider. From here on, `RequestList` means the possibly truncated version.

7. The storage provider checks if every `PostRequest` in the `RequestList` has a valid `PostSize`, `PostHash` and $\texttt{Sig}_{\texttt{request}}$.

8. If the `RequestList` contains a single root `PostRequest`, the storage provider checks if the `PostRequest` is consistent with its `StorageOffer`. Then the storage provider allocates a new chunk for the new thread.

9. If the `RequestList` contains one or more reply `PostRequest`s, the storage provider checks if all `PostRequest`s belong to the same thread. Then the storage provider checks if it is the primary storage provider for that thread. Finally, the storage provider checks if there is still enough space left in the remaining chunk to store all the incoming posts.

The next steps apply to both a root `RequestList` and a reply `RequestList`.

8. The storage provider aborts if any one of the previous checks fails. Otherwise, for each `PostRequest` in the `RequestList`, the storage provider saves the `PostHash` and `PostContent` to the chunk.

9. The storage provider generates $\texttt{Sig}_{\texttt{sp}}$ by signing the `RequestList` excluding the `PostContent` of each `PostRequest` contained within.

10. The storage provider sends $\texttt{Sig}_{\texttt{sp}}$ back to the miner as confirmation.

Verification Setup Protocol

After receiving the storage provider's confirmation, the miner must prepare the confirmed `RequestList` for verification.

1. The miner generates `PublicKey`, `SecretKey`, and a random `Name`.

2. The miner processes the confirmed `RequestList` and generates the authentication tags `AuthTags`.

3. The miner outsources `AuthTags` to the storage provider.

4. The miner combines the signed `RequestList` (excluding `PostContent`), the `PublicKey` and the `Name` to make a `Deal`.

5. The miner records the `Deal` in the new block.

Figure 3 summarizes the posting, matching and verification setup protocol.

Deal Verification Protocol

Each miner in the mining committee is responsible for verifying up to D (system parameter) deals. Suppose the committee has c miners, $c \leq K$, and suppose there are a active deals on the blockchain. Then the committee is responsible for verifying $d = \min(a, c \times D)$ deals and each miner is responsible d/c deals (if c does not divide d, the remainder is assigned to the miner with the first slot).

There is an integer *weight* associated with every active deal in the range $\{1...W\}$, where W is a system parameter. Every active deal is initialized with weight 1. During a mining round, the weight for an active deal increases by 1 if its verification fails, decreases by 1 if its verification succeeds and remains unchanged if it is not verified.

The d deals to be verified are sampled from the a active deals in a weighted uniform way. The sampling process is publicly verifiable because it uses a prescribed pseudorandom number generator (PRNG) seeded by the hash of the final block of the previous round.

A miner follows the deal verification protocol to verify the active deals that are assigned to it.

1. The miner generates a `Challenge` for each active deal and sends the challenge to the storage provider of that deal.

2. Upon receiving the `Challenge`, the storage provider generates a `Proof`, signs it, and sends it back to the miner.

3. If the miner receives the `Proof` within T_{ch} (system parameter) seconds, the miner combines it with the corresponding `Challenge` to generate a `Verification`. Otherwise, the miner generates a `Verification` containing the `Challenge` only.

4. The miner records all `Verification`s in the new block.

A deal verification succeeds if and only if the correspondng `Verification` contains a valid `Proof`.

Block Propagation Protocol

Nodes follow the block propagation protocol to distribute the latest blocks across the network. The protocol depends on two operations: `Ask(BlockHash, n)` and `Sync(blocks)`. `Ask` is a request to retrieve a chain of up to `n` blocks starting from but not including the block with `BlockHash`. `Sync` sends a chain of blocks to a designated receiving node, and each `Sync` operation can carry at most Z (system parameter) blocks.

A node new to a Timbre network needs to fetch the entire blockchain. It begins by locally broadcasting `Ask(Null, Z)`. After receiving a `Sync` from one of its peers, the node repeats `Ask(BlockHash, Z)` with updated `BlockHash` until it catches up with its peers.

A node also needs to fetch missing blocks after being offline for a period of time. It first finds the block common to its local main fork and the current network main fork that has the *greatest* height by doing a binary search on `BlockHash` with `Ask(BlockHash, 0)`. Then it repeats `Ask(BlockHash, Z)` with updated `BlockHash` until it catches up with its peers.

In addition to responding to `Ask`s, an up-to-date node should listen to new blocks that are broadcasted by miners. If a new block cannot be directly appended to its local blockchain, the node adds the block to its `BlockPool` of *pending* blocks. The node checks its `BlockPool` periodically for pending blocks that can be appended its local blockchain.

Analysis

In this section, we analyze different aspects of Timbre to shed light on the challenges in designing a system like Timbre and the compromises we made along the way. Our analysis is presented in a Q&A format to illustrate the motivation behind our design decisions.

Blockchain Design Analysis

- *How does Timbre deal with long-range blockchain attacks?* In contrast with Proof of Work (PoW) systems, Proof of Stake (PoS) schemes are vulnerable to *long-range attacks*. Because of significantly reduced computational expenses, PoS miners may recreate entire blockchain forks and earn system currency from mining on multiple chains over long periods of time (similar to the double spend attack). One proposed solution in the case of PoS is Ethereum's Slasher which uses economic and reputation "slashing" methods in order to discourage miners from dishonest forks. Fortunately, since DPoS blocks are mined by pre-determined miners (through voting and scheduling), *long-range attacks* are highly improbable since forking would require more committee members to work in agreement. However, a node could recreate a chain of blocks by using a different genesis block. This alternative chain is easily discardable since there is only one correct genesis block.

- *Why not tax everyone's balance at the beginning of a new epoch instead of resorting to a complete reset?* Uniform taxing is meaningless since it would simply maintain the power balances already established in the previous epoch. Progressive taxing is vulnerable to sybil attacks because nodes with high stakes can split their stake across multiple sybil nodes every time before a new epoch, essentially preserving their influence over epochs.

- *Why are old tokens redeemed block by block in the next epoch?* Without a limit on token redeeming rate, some nodes can earn back the allowed percentage of their old balances faster than others, thus recreating the long-term power imbalance issue the new epoch is supposed to address.

- *Why are storage providers not rewarded through inflation for offering free storage at epoch bootstrap time?* Rewarding storage providers with inflation for free storage deals may seem like a good way to bootstrap a new epoch. However, doing so makes

the system vulnerable to sybil attacks where storage providers pose as posters and submit large numbers of free `PostRequest`s, in the hope of being matched with those `PostRequest`s and gain inflation tokens.

- *Why can storage providers redeem old tokens by storing free storage?* At the start of a new epoch, we do not want miners to be the only nodes earning tokens in the system. Therefore, we allow storage providers to redeem old tokens by storing free posts in addition to paid ones. Sybil attacks have little effect because the system limits the redeeming rate.

- *Why must candidates have $\geq 1/s$ stake in their votes to be elected as miners?* In a situation where the mining committee has very few miners (e.g. in the early stage of a new forum when there is a lack of miner candidates), imposing a minimum stake in votes for elected miners prevents hostile take-overs by opportunistic nodes.

- *How do we deal with malicious miners?* Timbre assumes that the *majority* of the miner committee is honest; otherwise, DPoS consensus would break down. Still, a minority of miners could be motivated to eliminate competition from other candidates by censoring votes. They could also censor posts by dropping `PostRequest`s or favor certain storage providers (perhaps themselves) over others. However, since nodes are able to detect malicious behavior, as long as there is an honest majority, votes will still be correctly recorded and the malicious miners will be voted out.

Content Management Analysis

- *Why must a storage provider allocate a chunk for each thread?* By requiring the storage provider to allocate a chunk, miners are able to directly send future reply posts to the same storage provider until the chunk is full. It simplifies the process of matching a post to a storage provider, and it reduces *thread fragmentation* in the network. Since we assume a retrieval-by-thread forum access pattern, storing posts of the same thread at the same storage provider makes retrieval much more efficient.

- *What happens if a chunk of storage never gets filled?* This is indeed a possibility since only popular threads can generate enough replies to fill a chunk. Therefore, a significant portion of the storage provider's committed storage could be unused. Once the storage provider's committed storage reaches its total capacity, this inefficiency becomes an issue because although the storage provider still has plenty of unused storage, the unused storage cannot be used to allocate chunks for new deals. In this case, a conservative storage provider would simply wait for one of its deals to expire before accepting a new deal.

- *What if a storage provider "oversells" its storage?* A more aggressive storage provider that has a low used-storage-to-committed-storage ratio might oversell its storage to overcome the inefficiency. In fact, the most aggressive strategy is to keep accepting new deals until its used storage reaches its total capacity. However, such a storage provider will fail to store incoming reply posts of threads for which it is the primary storage provider. As a result, these threads are "dead" because our protocol requires miners to send reply posts to the primary storage provider.

- *Is it possible to punish a failing primary storage provider?* If a primary storage provider rejects incoming reply posts, then a miner may decide to match future root posts with other storage providers. Therefore, an overselling storage provider is risking its future income by maximizing its current income. Note that a primary storage provider may also fail for reasons other than overselling. For instance, we expect a significant portion of storage providers to be average forum users with commercial desktop or laptop computers. It is unlikely for these computers to remain powered on with stable internet connections all the time.

- *What happens when two miners make overlapping deals due to network delay?* Two deals *overlap* if they share at least one post in common. It is indeed possible for miners to make overlapping deals. Let A and B be two miners and suppose B is scheduled to produce a block after A does. If B does not receive A's block in time due to network delays, then it might produce a block with many deals that overlap with deals in the block that A produced. Once the network converges to a single longest fork, only one deal out of all its overlapping deals will remain valid, and the storage providers assigned to invalid deals may simply delete them.

- *Why does sampling in the deal verification protocol use with a prescribed PRNG seeded by block hash?* By requiring the sampling to be done with a prescribed PRNG with a public and unpredictable seed, the protocol prevents miners from favoring certain storage providers over others.

Reward Analysis

- *Why is a storage deal paid to its storage provider block by block?* We need a storage deal to be paid to its storage provider incrementally at each block in order to enforce storage for the complete duration of the deal. If a storage deal is paid in full to the storage provider when it is made, the system cannot deduct decibels from the storage provider even if it betrays the deal in the future, since otherwise node balances can never be fully determined and balance calculation becomes impossible.

- *Why does the miner earn the transaction fees from each storage deal and retrieval transaction?* The transaction fee incentivizes the miner to include as many deals and transactions in its block as possible within the block size limit, which helps increase system throughput.

- *Why does the miner also earn the transaction fee of storage deals block by block?* This way the system encourages the miner to choose reliable storage providers for forum content.

- *Why does the miner earn the verification reward for successful verifications only?* Although our Proof of Storage scheme ensures that storage is publicly verifiable, the miner is still capable of producing a *false-negative* `Verification` by ignoring a valid signed `Proof`. Since it is trivial to ignore valid signed `Proof` but impossible to forge a valid `Proof`[1], successful `Verifications` are essentially the only evidence that the miner has fulfilled its deal verification duty. Therefore, the system rewards the miner for successful verifications only.

[1] Unless the miner happens to also be the storage provider for the deal being verified. This issue will be discussed in section 6.

- *Why does the miner earn block rewards in addition to everything else?* Mining is critical in all blockchain systems, and the miner dedicates more resources in terms of computation power and bandwidth than any other node during its slot. Therefore, it is reasonable to provide block rewards to attract capable miners. In addition, block rewards are essential for currency bootstrapping at system bootstrap and at the start of each new epoch.

Future Work

Timbre is still a work in progress with many potential improvements to be made.

We have identified some vulnerabilities related to our current Proof of Storage scheme. First, the Proof of Storage scheme cannot verify replication of storage. As a result, the current Timbre system cannot enforce post replications for added redundancy. Moreover, the Proof of Storage scheme suffers from a potential backdoor attack in Timbre, where the miner assigns itself as a storage provider and is able to "pass" future verifications because it has the private key used to generate the proof parameters. For the first problem, Filecoin has proposed a *Proof of Replication* scheme [4] and open-sourced its implementation, but further research is required before we can adopt it in our system.

Statistics should also be collected once we have a running implementation. It will help us analyze how the system behaves under various conditions and also help nodes in the network make more intelligent decisions. Useful statistics include node uptime, network stability, mining quality and many more.

Finally, everything discussed thus far is related to Timbre's decentralized infrastructure. The implementation of Timbre infrastructure is under way, and the integration of application-level features is an important next step. Such features include a user interface and democratic content moderation, which will allow users to curate a more structured and opinionated view of the forum.

References

[1] J. Yuan and S. Yu, "Proofs of retrievability with public verifiability and constant communication cost in cloud," in *Proceedings of the 2013 International Workshop on Security in Cloud Computing*, ser. Cloud Computing '13. New York, NY, USA: ACM, 2013, pp. 19–26. [Online]. Available: http://doi.acm.org/10.1145/2484402.2484408

[2] D. Larimer, "DPoS Consensus Algorithm - The Missing White Paper," 2016. [Online]. Available: https://steemit.com/dpos/@dantheman/dpos-consensus-algorithm-this-missing-white-paper

[3] A. D. Peris, J. M. Hern'ndez, and E. Huedo, "Evaluation of the Broadcast Operation in Kademlia," *IEEE International Conference on Embedded Software and Systems*, 2012.

[4] J. Benet, D. Dalrymple, and N. Greco, "Proof of Replication Technical Report (WIP)," 2017.