DeSo - Decentralized Social Media

by Josh Painter, March 2022, v0.1

The Summary

This paper proposes a concept for a decentralized social network built on top of a simple decentralized data layer solution that can store key/value pairs. Users should be able to store their own social data locally and in a decentralized network of peer nodes instead of in a large company's remote centralized database.

The Problem

Users in the crypto world like to shout, "not your keys, not your coins!"

Over the last several years, we've seen this same principal applied to social media as well: not your social media site, not your social media! Those who have been censored by social media companies know this all too well. Each user should always be in control of their own social media data, and any other data that they own for that matter! Large, centralized organizations should not be able to limit speech simply because their corporate board disagrees with said speech.

The Solution

Each user will host two separate datastores known as Inbox and Outbox in a local datastore. This datastore will be synced with other users by using a decentralized peer-to-peer network. The details of the decentralized datastore are not important in this context, but it must at least support key/value pair type of data and it must be able to easily synchronize to peers.

Firstly, the Inbox is a stream of read-only content that gets synced from other followed users' Outboxes. The user may follow an unlimited number of Outboxes, but practical hardware limitations will require some additional complexity discussed later. Inbox stream data may be pruned according to the user's preferences.

Secondly, the Outbox is where the user will publish their own content, including re-posting content from other users that optionally includes the user's own content, like a "retweet" and a "quote tweet" respectively. Other users can then subscribe to or publish this user's Outbox so that the content flows into their Inbox and/or their Outbox, respectively, and so on. Outbox data should be stored by the user indefinitely on their local datastore.

The Details

Both the Inbox and the Outbox datastores will have the same structure: a simple list of key-value pairs. The key will be a SNOWFLAKE and the value will be a YAML document that describes the POST. Both data structures are very concise formats meant to save storage space. Using the SNOWFLAKE as the POST ID will also make data-import from centralized social media sites even easier for users.

There are two abstract types of POSTS: SINGLETON and CONTENT. A SINGLETON POST is a special kind of POST that is meant to be limited to exactly one of that kind of POST per Inbox. An example would be the user's profile data; the user could create (and keep updated) several SINGLETON POSTS that contain their user profile data - their name, banner image, avatar, email, public keys, DID, etc. Subscribers to this data store can the easily ask for these standard SINGLETON POSTS by name to get information about the owner and the datastore itself. Other SINGLETON POSTS may be used by 3rd party add-ins to store other standard state data for the user such as ownership of NFTs for display in the user's profile. These SINGLETON POSTS are mutable, but the history of their changes will remain in the datastore like all other data. If multiple posts of the same Singleton type exist, the most recent of these posts (by SNOWFLAKE) will be assumed to be the current and active one.

Unlike SINGLETON POSTS, a user may make any number of CONTENT POSTS. A CONTENT POST can be one of many CONTENT TYPES such as text, images, gifs, videos, Offer Files, etc. Each CONTENT TYPE will require a viewer application. The basic CONTENT TYPES above will be supported with viewers "out of the box" but 3rd party plugins could introduce new CONTENT TYPES and viewers as well.

Other special CONTENT TYPES of CONTENT POSTS are SUBSCRIBE, SUBSCRIBER, PUBLISH, PUBLISHER and BLOCK. When the user subscribes to, unsubscribes from, publishes, unpublishes or blocks another user, the appropriate CONTENT POST is created or deleted, respectively. When the user discovers that another user has subscribed to, unsubscribed from, published, unpublished or blocked them, the appropriate CONTENT POST is created or deleted, respectively. For more information on the difference between subscribe and publish, see Appendix A: Subscriber vs Publisher. For more information about Blocking, see Appendix B: Blocking Other Users and Censorship.

A user may take 4 actions on any CONTENT POST in their Inbox: REPOST, REPLY, UPVOTE, and DOWNVOTE. All these actions create respective REPOST, REPLY, UPVOTE, and DOWNVOTE CONTENT POSTS in the user's Outbox. Each of these types of CONTENT POSTS includes a link (the SNOWFLAKE) to the other CONTENT POST with which the user is interacting. The REPLY and REPOST actions may also include their own content from the user as well (the reply text and quote text for the REPLY and REPOST actions respectively).

Continued…

These SNOWFLAKE links are only resolved if the user has already subscribed to the corresponding author's datastore. A Content Viewer plugin could resolve these links by loading them from the original social media site's Web 2.0 API. If the user already follows that other user, the content will be resolved immediately from the local datastore.

> *Example: Bob follows Sally but not Bram. Sally upvotes and replies to a super-smart post made by Bram. Bob sees Sally's reply in his feed, but he doesn't see the content of the post to which she replied. He does see a little "Load Content" button instead, which he clicks. A custom Content Viewer plugin loads the post from the original social media site API. Once loaded, he reads the post and upvotes it. This Bram is a smart guy - Bob decides to follow him so he doesn't have to wait for Bram's content to load in the future!*

The User Interface

A simple user interface will be needed to make new posts and follow other users. To allow other users to follow the user, the user would publish their datastore's "id" and other users would copy/paste it into their client, thereby subscribing to it. Their client would then gossip out that id and ask any other nodes if they have that datastore for syncing.

The interface will look very similar to everyone's favorite 260-character site today. At the top, a "new post" button that will create a new post and add it to the user's database. The decentralized data layer will push it out to the user's subscribed followers from there.

Under that, the main feed. The interface will aggregate the user's posts with all other post datastores to which they are subscribed. The user may choose different Filters to apply to the list of aggregate posts. The simplest built-in Filter will show all aggregated, subscribed posts listed in reverse chronological order so the newest posts appear at the top. Each individual post will be rendered by the appropriate Viewer plugin depending on the CONTENT TYPE for that post. Viewers for simple text and graphics will be built in.

Third Party Plugins

The user interface will allow additional 3rd party plugins in the form of Filters, Viewers, CONTENT TYPEs and Importers.

- Filters: These open-source plug-in filtering and sorting algorithms could be installed at the user's discretion - use the "NSFW" view for when using company equipment, for example, and the app will not sync objectionable posts from follows. Use the "Popular" view to see the top posts ranked by upvotes. Use the "Offer Files" view to just show posts that include Offer Files. Use the "CATalog" view to see the decentralized CAT name/TAIL lookup datastore that uses knowledge of the crowd. These filters and views are *your* choices now instead of those of a centralized social media giant.
- Viewers: These open-source plugins would be responsible for rendering content from the post records. Viewers could be registered to handle one or more CONTENT TYPEs. Example 3rd party Viewers might enable consumption of video, audio, turn-based games, Offer Files, NFTs, etc. based on the CONTENT TYPE of the post.
- CONTENT TYPES: These open-source plugins would define new CONTENT TYPES. These CONTENT TYPES would be defined with a TBD schema definition. Filters and Plugins could either install their own CONTENT TYPES or work with existing CONTENT TYPES.
- Importers: These open-source plugins would import data from existing social media sites either for one-off initial large imports or ongoing synchronization.

Example: As mentioned in the previous paragraph, a new Filter, Content and Viewer plugin could be created by a 3rd party open-source developer that would provide a decentralized database of CAT names based on the TAIL. Today, several centralized Web 2.0 sites are responsible for this, but these plugins could work together to allow this directory to be decentralized and updated by the users themselves. A user could decide to publish their name for a specific TAIL by making a new post of CONTENT TYPE TAILNAME. This post would be propagated to all subscribers. The CATalog Filter plugin would only show the user posts of type TAILNAME. The Filter plugin would also allow the user to search by either TAIL or CAT name. The CATalog Viewer plugin would be responsible for rendering each item in the resulting list. This Viewer plugin could show a "probability" of the CAT name being associated to a specific TAIL by other members of the community, based on all the published TAILNAME CONTENT POSTs from other users that has been sync'd locally through subscriptions. In this way, CAT information is published through "knowledge of the crowd" instead of a centralized registry.

Import of and Interoperability with Existing Social Media Data

Because we use a SNOWFLAKE as the ID of the posts, we can easily import and integrate with Twitter. A user of Twitter can request an archive of their entire history of tweets, likes, etc. This archive includes the unique SNOWFLAKES for each piece of content. This content can be imported to a user's local outbox using the same SNOWFLAKES. This makes linking between imported content "just work" throughout the decentralized social network. Old imported re-tweets would still correctly link to the old tweets, assuming those were eventually imported to the decentralized network as well. New tweets made on Twitter could even be imported in real-time by a 3rd party plugin.

It is hoped that this will create more of a symbiotic relationship with existing centralized social media networks instead of an outright competitive relationship. Ideally, these centralized social media networks would become nothing more than nodes on the decentralized social media network, with no more rights or permissions than any other user.

A Note about Data Durability

It is important to note that decentralized datastores such as IPFS and Chia Data Layer do not intrinsically guarantee data durability. It will be important to remind users to backup their data, especially during the early days of their account when they don't have very many subscribers or publishers.

The more popular social media stores would naturally be the most protected against data loss because they will have the most subscribers and publishers and therefore the most unique copies on unique nodes. But users would still be encouraged to back up their local posts datastore to prepare for the inevitable and unfortunately sad scenario of a user having a lot of posts but no active followers acting as a backup. :'(This backup feature is not part of this proposal but is expected to be supplied by another future wallet add-in.

Why Chia?

Most attempts at new decentralized social media sites have failed. In fact, most attempts at new social sites of any sort have failed! New entrants to this market face the almost insurmountable "chicken or the egg" problem. People won't join if their friends aren't already there, and likewise. This gives current social media behemoths a near-monopoly on this user content and data and makes it very hard for users to leave the "walled gardens."

A social network built on top of Chia, however, would already have hundreds of thousands of users "baked in" who are already incentivized the run the Chia wallet, through farming and/or transacting. These users are also the most valuable kinds of users: early adopters who are excited and hungry for Chia to grow. They would spread news of this new decentralized network on all their existing social platforms and work tirelessly to convert users!

Assumptions about Chia Data Layer

- Subscribing to a datastore is as simple as copy/pasting an ID, like adding a new CAT into your wallet by Name/TAIL
    - Looks like current implementation requires datastore ID, IP address and port
- DataLayer can handle an unlimited number of local data stores
    - A social media account can follow tens or even hundreds of thousands of other social media accounts, so it is important to understand DataLayer's scalability targets. Another option is to introduce open-source aggregator nodes – instead of subscribing to individual users, a user might subscribe to one or more aggregator nodes that aggregate posts from lots of different user databases into one (still decentralized) datastore. The different aggregators' popularity would likely be based on the quality of curation of those posts. See Appendix A: Subscriber vs Publisher for more formalized definition.
- DataLayer lets only the creator of the datastore write to it
    - All others are subscribers with read-only access
- DataLayer will eventually support Merkle-tree lookups from non-subscribed datastores
    - This would be great to quickly load posts from linked datastores BUT it might make more sense to leave this feature out to incentivize users to publish other users and duplicate data as much as possible
- DataLayer will eventually support backup options, so users have an option to restore data in case nobody else subscribes to or publishes them
    - This also assumes that a local writable DataLayer datastore can be recreated from a remote read-only DataLayer datastore (assuming the original creator provides their private key).

Appendix A: Subscriber vs Publisher

The difference between a Subscriber and a Publisher is subtle but important and solves an interesting technical problem. One of the largest problems in any social media network is that of scale: some users follow (subscribe to) tens of thousands or even hundreds of thousands of other users. Other users have millions of subscribers. In a true peer-to-peer decentralized network it would be untenable for each user to connect directly to another user's datastore. The user that has millions of subscribers would not be able to sustain millions of incoming connections just as the user that follows tens of thousands of other users would not be able to reliably sustain connections to all those other users.

Ideally, aggregators would pop-up on the network and users could connect to those instead. Aggregators will combine and publish content from different users and a follower of the aggregator would automatically receive the content of all users for which the aggregator itself publishes. A "tech" aggregator could publish 100 tech accounts – or even other "tech" aggregators! A user could choose to follow certain aggregators based on their individual curation choices. In this way, a user could receive a massive amount of content from hundreds of thousands of different sources by choosing a few top-level well-known aggregators to follow.

We will now rename "aggregator" to "publisher." A publisher is very much like a subscriber, but when a user decides to publish another user, they are making a commitment to store that published user's content along with their own, so that any subscribers also receive that other published content. In effect, by publishing another user, the user will automatically re-post the other user's content into their own Outbox, therefore pushing the content out to their subscribers (and publishers) as well. Here is the previous paragraph above, but now we've substituted "aggregator" with "publisher."

> *Ideally, publishers would pop-up on the network and users could connect to those instead. Publishers will combine and publish content from different users and a follower of the publisher would automatically receive the content of all users for which the publisher itself publishes. A "tech" publisher could publish 100 tech accounts – or even other "tech" publishers! A user could choose to follow certain publishers based on their individual curation choices. In this way, a user could receive a massive amount of content from hundreds of thousands of different sources by choosing a few top-level well-known publishers to follow.*

Of course, the user could then choose just a few users within that large set of users to publish, and then the user's subscribers will now receive that published content as well!

Appendix B: Blocking Other Users and Censorship

The concept of blocking another user usually means that blocked user can no longer see the user's posts and vice versa. However, because datastores are public data, blocking a user here just means that the blocked user's content is filtered out of the user's feed.

Consider that on the top social media sites, content is public by default and a blocked user could easily log out and back in as a different account to see the content. Blocking is usually more useful for filtering out the offending user's content from the user's feed instead of stopping the offending user from accessing the user's content.

Block CONTENT POSTs should always be respected by plugins when possible. Sometimes, it might be unavoidable to show content from a blocked user. Usually, these exceptions involve historical transactions or other CONTENT POSTs that are needed by plugins to continue functioning. Plugins should still strive to respect the user's Block instructions for any opinionated content that the blocked user might produce, including the user's avatar image, the user's username, etc. These Blocked bits of content should still be accessible behind a link or button in the viewer, should the user still choose to view the blocked content.

Most importantly, the concept of blocking should always be a personal decision by the user, and it should only affect that user's view of the social network. One user (or corporation) shall not and cannot stop another user from posting, but one user doesn't have to listen to another user either.

To sum up: censorship should never extend beyond an individual's personal choice of the content they themselves consume. Those choices shall be respected by the social network and the plugins themselves where possible.