Decentralized identifiers as a means to coordinate peer-to-peer services

An exploration of DIDs in a thread-based IPFS network

Sander Pick*, Carson Farmer*†, Ignacio Hagopian*, Andrew Hill*, Aaron Sutula*, Ben Wilson*
*Textile.io †carson@textile.io

***Abstract*—This is the abstract.**

It consists of two paragraphs.

I. Introduction

A major promise of the decentralized web is putting ownership back into the hands of users by shifting publishing, discovery, and services from a few core providers, onto the "edge" and into the hands of users/creators. True networks of untrusted peers that can coordinate [1] to provide important peer-to-peer (p2p) infrastructure and services.

Core components of the decentralized (or distributed) web, such as storage, have received a great deal of attention, including projects such as the Interplanetary File System (IPFS) [2], Filecoin [3], Sia [4], and Storj [5]. These projects are demonstrating that distributed, (incentivized) storage is possible, and even profitable. However, *services* such as those traditionally found in centralized systems via open (or closed) application programing interfaces (APIs) are less common in p2p systems.

The "Thread Network" is a a proof-of-concept network that aims to address this shortcoming. The proposed system consists of a network of peers working to manage the discovery and coordination of an unbounded set of p2p services. In this paper, we focus on a core piece of this proposed system: service discovery. We propose the use of decentralized identifiers (DID) as a means to coordinate peer-to-peer services.

II. Background

A. Decentralized Identifiers

A decetralized identifier, or DID, is a string identifier of a *subject*, controlled by a *controller*. A DID might be used to encode a reference to an Ethereum address [6] on the Ethereum buterin2014next, [7] Mainnet, identify a resource on a network of IoT (Internet of Things) devices, or even represent a unique "identity" such as a user or organization [8]. For DIDs to be useful, they must be "resolvable" without reliance on a centralized network component. Fig. 1 shows the interactions between DID components.

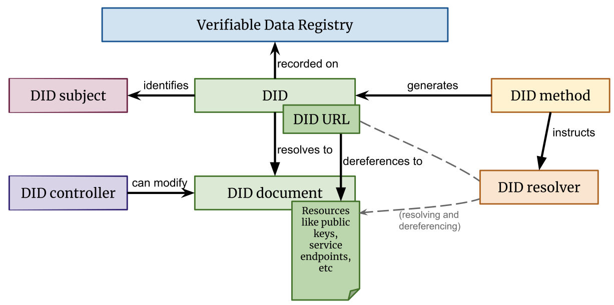

Figure 1: The basic components of DID architecture. Source: [9] sec 1.3

The *Verifiable Data Registry* in fig. 1 could be a blockchain or peer-to-peer network. The meaning of *verification* differs between DID implementations – some provide a high level of verification through blockchain transactions, while others rely purely on the assumption that the majority of peers are *good actors*, e.g., IPID (an IPNS-based DID) or Ceramic.

A DID can identify any actor or structure in the network (the *subject*), and should resolve to a *document* that is accessible from anywhere in the network. Formally, a DID document describes the *subject*, and the document is controlled by one or more *controllers*. In short, DIDs point to relationships between the components of a decentralized network. What can this component do? Who/what can do it? Who/what dictates who can do it? See [9] for the DID specification.

B. Threads

A thread is topic-based collection of single-writer logs. A collection of logs represent updates to the "state" of an object (or dataset). The basic units of a thread – logs and records – provide a framework for creating, storing, and transmiting data in a p2p distributed network. The thread protocol is outline in detail in the threads whitepaper [10].

A log within a thread is essentially a set of cryptographically linked (i.e., hash-linked) records, that form a specific type of Merkle-DAG (directed acyclic graph) that represents a purely functional [11] and authenticated [12] (i.e., immutable) singly-linked list. The key insight here is that, assuming two peers have received all of the same updates to a thread, they will deterministically arrive at the same thread structure, and that thread structure can be summarized by the (set of) hash(es) of the head(s) of the underlying log(s).

III. Thread DIDs

In practice, a thread *network* may have multiple actors and structures (subjects) that can be described by DID documents. Subjects include any of the following entities:

- Peers *Peers* (e.g., `did:p2p:foo`) can offer network services, such as thread "hosting," pinning services (i.e., IPFS, IPNS), Filecoin anchoring, database abstractions, API services, and more. A thread peer's DID is derived from its embedded networking host's key (which in practice is a `libp2p` (https://libp2p.io/) peer]).
- Users A *User* (e.g., `did:key:foo`, `did:3:foo`, `did:ethr:foo`, etc.) is any external identity that represents a network user, and that may interact with the network *via* a Peer. These may be identified by any verifiable DID.
- Thread*Thread DID* (e.g., `did:thread:id`) documents contain verification methods for all valid log signers. Other thread info such as the *log head*, *log metadata*, and *thread encryption keys* are *not* stored in the DID document, as this information is only needed by peers that are sharing a thread, and can be more efficiently exchanged using the thread protocol [10] directly (vs. a global document registry). *Log addresses* are referenced as *service endpoints*, as defined in [9] sec 5.4.

By identifying a thread as a global resource, any peer can determine the following from its DID:

1. Who can write to the thread?
2. Who can read from the thread?
3. Where can the thread be bootstrapped from?
4. Who controls (1), (2), and (3)?

It is important to note that a thread DID is an *identifier*, not an *identity* in the usual sense. To illustrate, if a thread is a resource that a peer on the network is attempting to identify, i.e., the thread itself *is the subject*, and the thread document is a *representation of the subject*, then the DID *subject is not the controller* [9]. The controller is one or more identity-based DIDs, e.g., `did:key:foo`, `did:ethr:foo`, `did:3:foo`, etc.

A (proposed) thread DID document is structured as in lst. 1. In this example, the controller (`did:key:foo`) is defined by a key-based DID [13]. The controller is able to modify the thread DID document. Additionally, both `did:key:foo"` and `did:key:bar"` can authenticate as `did:thread:id`, meaning . . . . Finally, the thread DID document in lst. 1 also specifies two `serviceEndpoints` that outlines which peers (`/p2p/peer-id-1` and `/p2p/peer-id-2`) are able to bootstrap the thread.

Listing 1 Proposed thread DID Document structure.

```
{
  "@context": "https://www.w3.org/ns/did/v1",
  "id": "did:thread:id",
  "controller": "did:key:foo",
  "authentication": [
    {
      "id": "did:key:foo#keys-1",
      "type": "Ed25519VerificationKey2018",
      "controller": "did:key:foo",
      "publicKeyBase58": "..."
    },
    {
      "id": "did:key:bar#keys-1",
      "type": "Ed25519VerificationKey2018",
      "controller": "did:key:bar",
      "publicKeyBase58": "..."
    }
  ],
  "service": [
    {
      "id": "did:thread:<id>#peer-id-1",
      "type": "threadService",
      "serviceEndpoint": "/p2p/peer-id-1",
      "serviceProtocol": "/thread/0.0.1"
    },
    {
      "id": "did:thread:<id>#peer-id-2",
      "type": "threadService",
      "serviceEndpoint": "/p2p/peer-id-2",
      "serviceProtocol": "/thread/0.0.1"
    }
  ]
}
```

IV. Method Definition

The crux of any DID implementation is defining it's *DID method*. The DID specification defines a *method* as a "means to implement this specification on different verifiable data registries" [9] (sec 8). In other words, new DID methods should be specified such that different implementations of the same DID method remain interoperable.

Here we define a DID method specification for threads which is composed of a *method scheme* (see [9] sec 3.1) and *operations* (sec 8.2). Operations specify how a DID document is created, how to read/verify a document, as well as how a DID controller can update or even deactivate a DID document. The method scheme defines the structure of the DID implementation's string identifier(s).

A. Method Scheme

A thread DID method scheme prefixes the unique identifier for a thread (see [10] sec 2.2), with the globally unique `did:thread:` namespace, such that a thread DID becomes `did:thread:<thread-id>`, where `thread-id` is defined as:

$$\underbrace{\texttt{0x62}}_{\text{Multibase}}\ \overbrace{\texttt{0x01}}^{\text{Version}}\ \underbrace{\texttt{0x55}}_{\text{Variant}}\ \overbrace{\texttt{0x539bc...a4546e}}^{\text{Random Component}}$$

This produces a string identifier of the form: `"did:thread:bafk6npbyp...6mfuhoe6iesr"`

B. Method Operations

As mention perviously, DID methods define a set of operations that can be performed on/with DID documents. DID implementations often use a smart contract [14] on a blockchain like Ethereum to model the global data registry, and to implement these operations. For the implementation proposed here, we assume that documents are stored on the Filecoin blockchain as non-fungible tokens (NFTs). Filecoin does not yet support NFTs, but there is an informal proposal to create a new *actor* type for them.

In our initial proof-of-concept network, a non-consensus driven global data registry is used, based on a p2p "gossip" protocol. In practice, this is implemented using libp2p's gossip-sub implementation. This registry provides weak consensus, along the lines of IPNS-over-pubsub [15].

The process of getting a DID subject's document from the verifiable data registry is called *resolution*. Any peer can resolve any document by querying the on-chain NFT representing the subject, or in the shoter-term case, by posting queries to the associated pubsub channel.

V. Architecture

The way in which a DID method is leveraged is entirely up to the network itself. Here we outline some system requirements and walk through common network operations.

A. Vanilla Network

First, let's consider a network of completely open peers. These can be local (all on the same machine) or remote (distributed across a network). Open here means *no identity authorization*, such that anyone can create/add and read/write to threads.

A minimal set of requirements for this type of network to operate includes allowing external identities to leverage a peer (local or otherwise reachable by the user) to create threads. Once a thread is created, it can be considered globally available, i.e., thread peers can do work on behalf of other thread peers.

In practice, only peers that have been used to read/write to a given thread will follow said thread, and as with most other operations, any peer can be used to delete a thread. A simplified representation of a "vanilla" thread network such as this is shown in fig. 2. In this case, the network consistst of $n = 2$ peers interacting with an external identity A that is requesting operations.

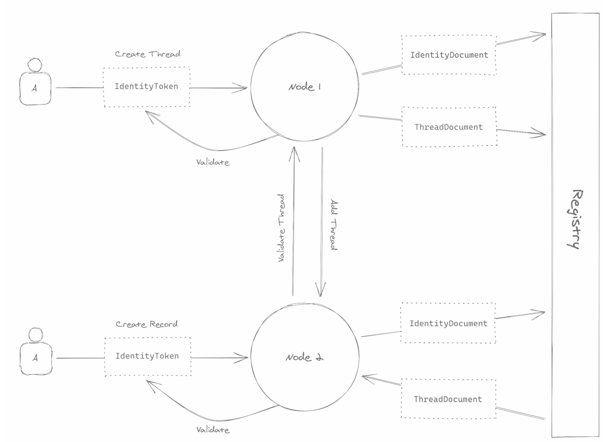

Figure 2: A vanilla thread network ($n = 2$) showing an external identity (A) creating a thread on one peer and writing to it from another peer.

B. *Services*

In our hypothetical vanilla network network, one of the peers has a *trusted relationship* with a service. Peers can advertise their services using the `services` DID field. For example, consider a hypothetical web-hook service that allows users to add web-hooks to a thread. Every time the thread receives an update, the web-hook fires on the user-defined endpoint. The peer's DID document includes the service information:

Listing 2 Thread DID Document with service information.

```
...
"services": [
    {
      "id": "did:key:peer-id#threads",
      "type": "threadService",
      "serviceEndpoint": "/p2p/peer-id",
      "serviceProtocol": "/thread/0.0.1"
    },
    {
      "id": "did:key:peer-id#webhooks",
      "type": "threadWebHookService",
      "serviceEndpoint": "/p2p/peer-id",
      "serviceProtocol": "/thread/0.0.1",
      "cost": {
        "hook": {
          "amount": "xx nanoFIL",
          "currency": "FIL"
        },
        "hit": {
          "amount": "xx nanoFIL",
          "currency": "FIL"
        }
      }
]
...
```

Services in this context are relatively flexible, and because "trust" is handled in separate layers of the system (i.e., via blockchain transactions, payment services, contracts, etc.), services can define their own authorization patterns. Services may be free, have free quotas, only be open to some users, etc.

To work with the Thread Network, a peer's services must be discoverable from their DID document. Services must also be self-describing via the DID service `type` , `serviceEndpoint`, and `description` fields. The community can maintain a list of available services by type to further aide in service discovery.

As alluded to above, service discovery is a key feature of the proposed system. Peers/users must be able to discover services on the network, without relying on a single "indexer" or centralized API. In this initial proof-of-concept, service discovery is handled via peer gossip/pubsub: Peers are used to request service types across the whole network via libp2p pubsub messages. Matching host peers respond directly to the caller with a verifiable service description.

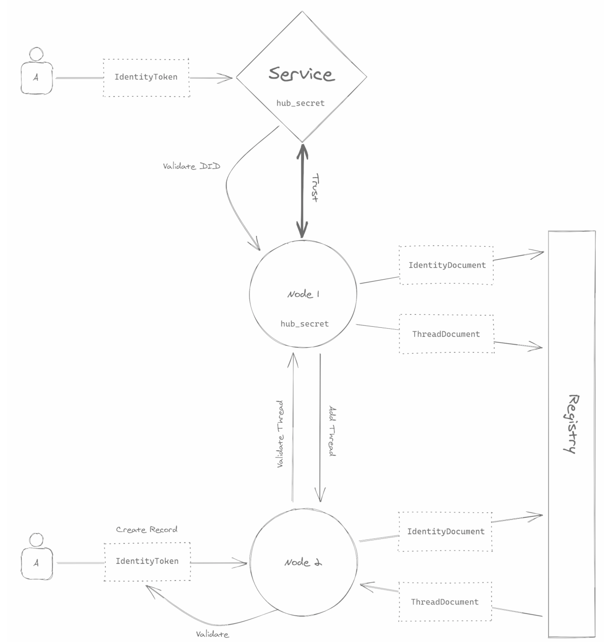

Figure 3: A service-enabled thread network ($n = 2$) with a hypothetical web-hook service. An external identity (A) has used the service to add a web-hook to a thread and then writes to the thread from another peer.

At a minimum, a peer will advertise it's own `libp2p` thread API. One of the key goals for the Threads Network is to enable (and encourage) external tool integration. By this we mean the ability to create and push data to the threads network via existing (external) tools such as databases (e.g., MongoDB, PostgreSQL, Redis), chat and messaging protocols (e.g., Matrix and ActivityPub clients), rich text editors (e.g., Quill, Slate, CodeMirror), etc. Here are some examples of additional services a peer might offer:

- Buckets
  - Mutable filesystem API
  - Pinning API
- A `go-datastore` interface (key-value store)
- Web-hooks
- Filecoin
  - Thread anchoring
  - Bucket archiving
  - Deal retrievals

- Databases
  - MongoDB (e.g., direct connection URI)
  - PostgreSQL
- Media encoding
- etc.

The diagram in fig. 4 shows how a pinning service (via Buckets) and Filecoin can be orchestrated as peer services.

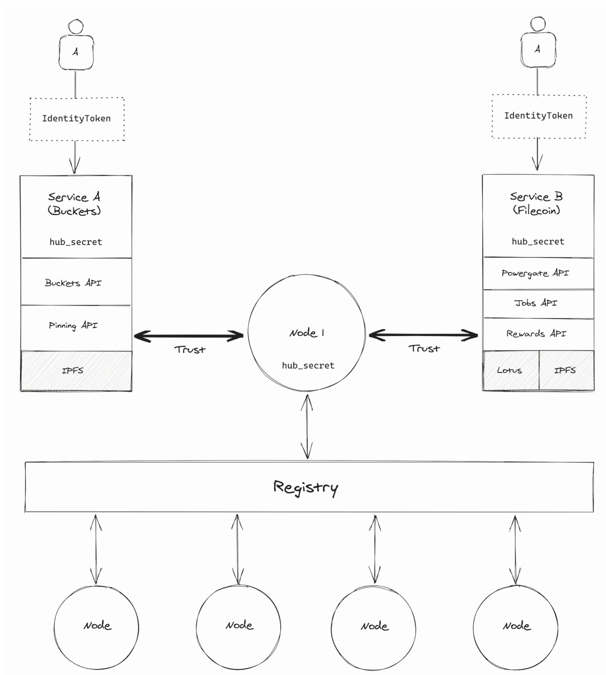

Figure 4: Buckets and Filecoin can be considered network services.

In addition to exposing more "traditional" web2 APIs (REST, gRPC, etc) as services, it is possible to expose these APIs over p2p protocols. Projects such as libp2p-http, allow peers to serve HTTP endpoints and make HTTP requests through `libp2p` using Go's standard "http" and "net" stack. This provides a simple on-ramp for web2 developers to expose their APIs over the threads p2p network. Couple service provision and remote database access with the robust authentication and globally identifiable assets afforded by thread DIDs, and we now have a very clear path to onboarding web2 developers (and data) to the decentralized web.

VI. Conclusions

Decentralized identifiers (DIDs) enable cross-protocol/blockchain interactions by defining a common interface to retrieve and validate entities, such as users and data. A DID-driven threads network offers an exciting opportunity to expose p2p services as first-class components of a p2p system.

Threads and IPFS/Filecoin benefit from a native DID in numerous ways. Firstly, DIDs provide globally unique namespaces for resources, such that thread IDs act as identifiers to network-wide resources. Secondly, adopting a standardized system for resource identification increases ecosystem(s) interoperability, facilitating integration with external DID methods available on systems such as Filecoin or Ethereum, as well as a range of identity solutions (e.g., Ceramic/IDX). Thirdly, globally unique identifiers pave the way for *distributed authorization* to network resources, because access control mechanisms are globally and unambiguously defined. This leads to a net increase in the decentralization of network services.

An additional key benefit of the proposed system includes the provision of "native" discovery mechanisms. Network peers can do work on each others' behalf since any peer is capable of discovering and resolving thread DID namespaces. Additionally, since all operations on a thread lead to deterministic, content-addressable updates, any peer on the network can participate in the update of a thread to which they have access.

There are additional motivators for exposing a DID-friendly threads specification. One such motivator is providing "crypto-native" access to off-chain data. Decentralized app developers can enable their users to directly leverage local crypto wallets to interact with thread-enabled apps. This includes Filecoin integration. Threads then become append-only logs that can be anchored to Filecoin manually or at some frequency. Buckets (derivatives of threads) can expose a mutable filesystem and pinning API that can be similarly archived to Filecoin. In this sense, DID-based threads can become a Filecoin "Layer 2" protocol.

References

[1] C. Barabas, N. Narula, and E. Zuckerman, "Defending internet freedom through decentralization: Back to the future," *The Center for Civic Media & The Digital Currency Initiative MIT Media Lab*, 2017.

[2] J. Benet, "IPFS: Content addressed, versioned, P2p file system," *arXiv preprint arXiv:1407.3561*, vol. (Draft 3), 2014.

[3] Protocol Labs, "Filecoin: A Decentralized Storage Network," Jul. 19, 2017.

[4] D. Vorick and L. Champine, "Sia: Simple decentralized storage," *Nebulous Inc*, 2014.

[5] S. Wilkinson, T. Boshevski, J. Brandoff, and V. Buterin, "Storj a peer-to-peer cloud storage network," *Storj Labs, Inc.*, 2014.

[6] C. AG, "Ethr DID resolver," *GitHub repository*. https://github.com/decentralized-identity/ethr-did-resolver; GitHub, 2021.

[7] G. Wood and others, "Ethereum: A secure decentralised generalised transaction ledger," *Ethereum project yellow paper*, vol. 151, no. 2014, pp. 1–32, 2014.

[8] C. Network, "Ceramic protocol specification," *Protocol Specification*. https://github.com/ceramicnetwork/ceramic; Ceramic, 2021.

[9] D. Reed, M. Sporny, D. Longley, C. Allen, R. Grant, and M. Sabadello, "Decentralized identifiers (DIDs) v1.0: Core architecture, data model, and representations," *W3C Working Draft*, 2020.

[10] S. Pick, C. Farmer, A. Sutula, I. Hagopian, I. Gozalishvili, and A. Hill, "A protocol & event-sourced database for decentralized user-siloed data," *Threads Whitepaper*, vol. v1.6, 2020.

[11] C. Okasaki, *Purely functional data structures*. Cambridge University Press, 1999.

[12] R. Tamassia, "Authenticated data structures," in *European symposium on algorithms*, 2003, pp. 2–5.

[13] D. Longley, D. Zagidulin, and M. Sporny, "The did:key method v0.7: A DID method for static cryptographic keys," *W3C Unofficial Draft*, 2021.

[14] V. Buterin and others, "A next-generation smart contract and decentralized application platform," *white paper*, vol. 3, no. 37, 2014.

[15] V. Santos and S. Allen, "IPNS – interplanetary naming system," *Protocol Specification*. https://github.com/ipfs/specs/blob/master/IPNS.md; IPFS, 2020.