Murmurations

A distributed data sharing protocol

White Paper v1.0 - April 2024
Geoff Turk & Oliver Sylvester-Bradley

Introduction

> *There is nothing more difficult to take in hand, more perilous to conduct, or more uncertain in its success, than to take the lead in the introduction of a new order of things. — Niccolò Machiavelli*

Murmurations is a distributed data sharing protocol. It describes a method for coordinating the storage, indexing and retrieval of data that enables a decentralized database. The ultimate goal of Murmurations is to facilitate collaboration at scale by enabling interoperable data sharing across platforms and between networks, while providing individuals and other data creators control over their data.

Overview

What is a database?

> *A database is a collection of data organized for rapid search and retrieval by a computer. [ref]*
>
> *A database is a systematic way of storing information to be accessed, analyzed, transformed, updated and moved with efficiency. [ref]*

A database performs two key functions that make large-scale information management possible:

1. It stores information following a defined format
    - **`name`**, **`street_address`**, **`city`** as strings of text, **`country`** as a 2 letter code, etc.
2. It enables fast search and retrieval of information
    - Find every node in Canada (CA) with the word "brewery" in its **`name`**

What are the advantages and disadvantages of a decentralized database?

Advantages

A decentralized database offers several advantages over the centralized database approach that is the model of nearly every database we interact with today.

Decentralized Databases	Centralized Databases
Enable custody of our data to parties we trust, or to manage our data ourselves using the right tools.	Entrust a small number of remote entities to hold our data, even if they don't prioritize our best interests.
Data commons - Increased interoperability of data and ability to choose where it can be used.	Siloed data - limited interoperability and data portability (platform lock-in).
Enable data owners to make the same data available for multiple uses, restrict access to their data, or share it widely when appropriate.	Data duplication - how many websites have we typed our address into and how long would it take to make a change in each of them?

First of all, decentralized databases allow us more granular control over who we trust to hold our data. We can decide where we want to store various information about ourselves, our projects, possessions, economic interactions, etc. As long as we follow the protocol for data storage, the location where it is stored can be anywhere on the internet.

Second, this flexibility of storage location discourages data silos and platform lock-in. The fact that data can be stored anywhere the data owner/creator chooses means that they do not have to rely exclusively on the tools of a single provider when it comes to accessing, searching, sharing and retrieving their data. A true data commons becomes achievable when data storage and searching is based on an open protocol.

And finally, data duplication as well as redundant and outdated data are no longer an issue like with centralized databases. Data creators store their data in one place and those who wish and are authorized to access it can then pull the latest changes to that data whenever it is updated (see the section Source data & aggregated data below).

Disadvantages

Decentralized Databases	Centralized Databases
Indexes and data tables are distributed across the network so response time is highly variable, since data is fetched from multiple hosts.	Offer faster response times because the database is hosted on one server or a cluster of servers, usually in the same physical location.
Variable availability - depends on the overall reliability of nodes and indexes.	Can offer high availability with enough investment in infrastructure.

By its very nature of being decentralized, we can not expect such a database to be as fast as a conventional centralized database. But once the relevant data is collected during aggregation, all the performance characteristics of a conventional database can then be leveraged by an aggregator (see Aggregators below).

How is the Murmurations decentralized database structured?

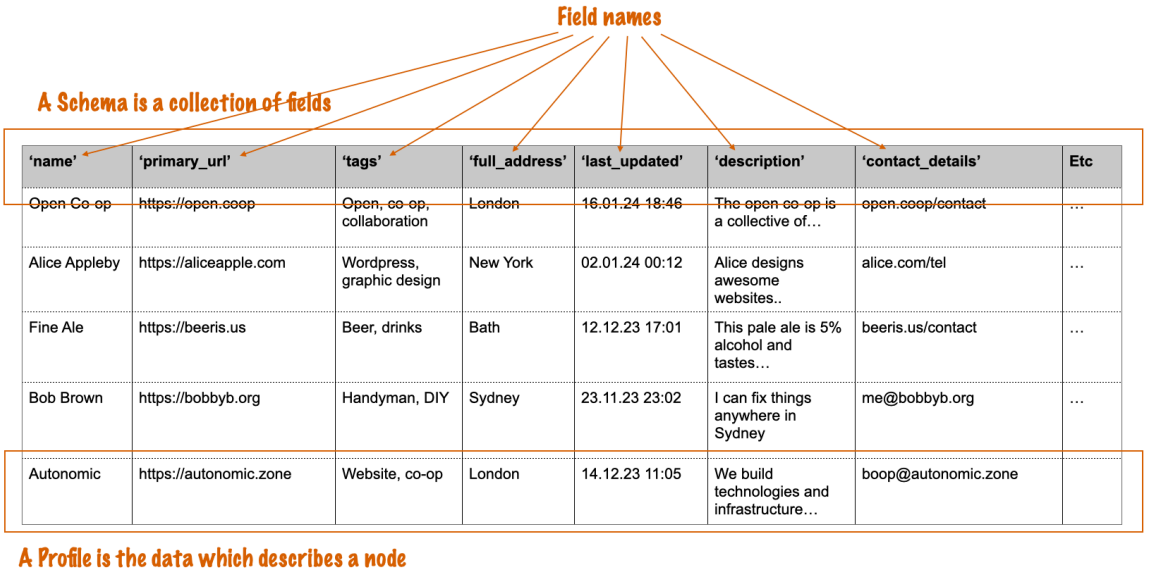
Recall the two key functions of a database described earlier:

1. It stores information following a defined format
2. It enables fast search and retrieval of information

A centralized database takes on both of these functions entirely on its own. The information is sorted into columns (e.g., `name`, `street_address`, `country`, etc.) and stored in tables, with each row in the table containing the information about a data subject. A centralized database also has indexes which record the location of data in certain columns (e.g., `country`) in order to quickly perform queries to retrieve a subset of the table's data.

Murmurations splits up the functions of a centralized database and shares the work across several different entities:

- Each 'row' in a decentralized database 'table' is a record stored independently on the internet (a “profile”), preferably by the data owner ("node") or a trusted party.
- The 'columns' (what we call "fields") define the types of data in the decentralized database 'table'. Collections of fields are defined by "schemas". We refer to the schema hosting service as a “library”.
- A subset of the fields from every profile are recorded in a service called an “index” so that profiles can be quickly searched and then retrieved, just like in a centralized database.
- Multiple libraries and indexes can be operated by different parties, all conforming to the same protocol to ensure interoperability of profile data.



‘name’	‘primary_url’	‘tags’	‘full_address’	‘last_updated’	‘description’	‘contact_details’	Etc
Open Co-op	https://open.coop	Open, co-op, collaboration	London	16.01.24 18:46	The open co-op is a collective of…	open.coop/contact	…
Alice Appleby	https://aliceapple.com	Wordpress, graphic design	New York	02.01.24 00:12	Alice designs awesome websites..	alice.com/tel	…
Fine Ale	https://beeris.us	Beer, drinks	Bath	12.12.23 17:01	This pale ale is 5% alcohol and tastes…	beeris.us/contact	…
Bob Brown	https://bobbyb.org	Handyman, DIY	Sydney	23.11.23 23:02	I can fix things anywhere in Sydney	me@bobbyb.org	…
Autonomic	https://autonomic.zone	Website, co-op	London	14.12.23 11:05	We build technologies and infrastructure…	boop@autonomic.zone	

Components

The following components and their interactions make up the Murmurations distributed data sharing protocol.

Nodes

A node (as defined in relation to a graph database) is an entity (person, project, organization, event, etc.) that has a profile which is registered in the Murmurations index.

Ideally nodes are managed by the data owner for maximum data authority and reliability. For example, data describing XYZ Org hosted at `xyz.org` is more likely to be accurate and authoritative than data describing XYZ Org which is hosted at `another-site.com/xyz_org`.

Nodes generate source data. Each node is recognized as the implicit owner of the data about it. When an organization states its name, mission and founding date, that data is owned by that named organization. When an individual posts a 3 sentence thought about a topic, that data is owned by that individual.

It must be highlighted that most source data today is not controlled (or even completely owned) by its data source. Instead, source data is often hosted by third parties and sometimes the link of ownership of that data and its allowed use is weakened for the data source by implicit contracts created between the data source and the third party data host. All too often source data is reused by the third party in multiple ways for marketing, profiling and other purposes.

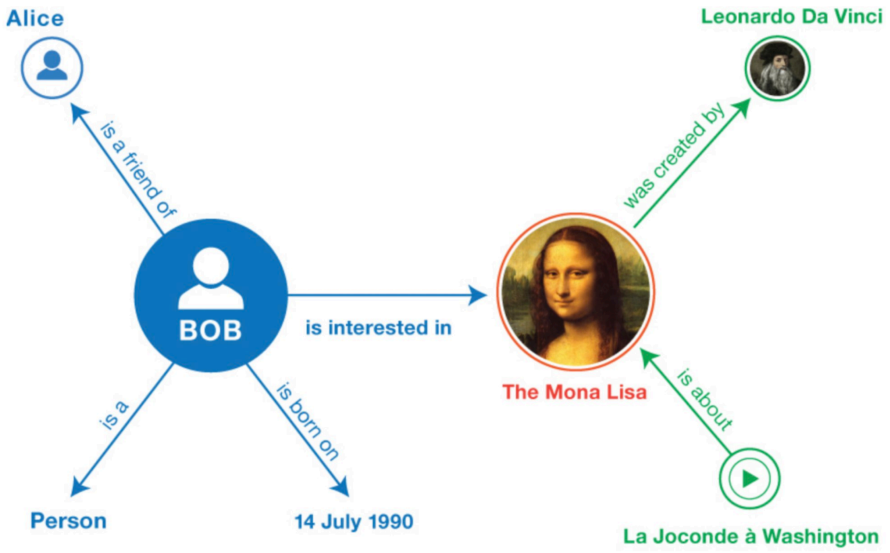
In Murmurations data sources can host their data themselves (e.g., by posting it to their websites) or use a third-party service that hosts their data for them. A working example of a data hosting service is at <https://test-tools.murmurations.network/profile-generator> in the Murmurations test network, and the code is maintained at <https://github.com/MurmurationsNetwork/MurmurationsTools>.

An example of a self-hosted node profile can be found here: <http://open.coop/open.json>

Relationships mapping

Nodes are not restricted to posting information only about themselves; they can also make statements about their relationships to other nodes; thus Murmurations enables the semantic web. These relationships can be captured using a *semantic triple*, which "is a sequence of three entities that codifies a statement about semantic data in the form of `subject→predicate→object` expressions (e.g., 'Bob is 33', or 'Bob is friends with Alice')." [ref]

Triples can be combined in a myriad of ways to express simple and complex relationships between nodes.



We have implemented the `relationships` triple in Murmurations as follows:

1. The node described in the profile is presumed to be the subject of the triple (e.g., Bob in the diagram above), thus the implied `subject_url` is in fact the node's `primary_url`.
2. A list (array) of relationships (objects) is then described with the relevant predicates and objects, for example:

```
"relationships": [
    {
      "predicate_url": "https://schema.org/contributor",
      "object_url": "https://murmurations.network/"
    },
    {
      "predicate_url": "https://schema.org/memberOf",
      "object_url": "https://www.collaborative.tech/"
    }
  ]
```

The relationships field in Murmurations is defined here: <https://github.com/MurmurationsNetwork/MurmurationsLibrary/blob/main/fields/relationships.json>

This is just one possible implementation of semantic triples in Murmurations, which is influenced by the resource definition framework (RDF) graph model. Schema creators may define others that better meet their use cases and then host them in their own libraries.

Libraries

A library is a service that records the definitions of data, specifically:

1. The fields that define a unit of data (e.g., name, country, etc.)

2. The schemas that are composed of fields (e.g., the Murmurations Organizations schema)

You can think of schemas as files that define the names and validation parameters of the first row in a spreadsheet of data. In other words, a schema defines the structure of a table in a conventional database.

Just as public libraries are distributed around the world yet share many common books, Murmurations libraries should also be widely distributed and host common fields, and schemas composed from those shared fields. Index operators and even aggregators can replicate the Murmurations Library repository (use the common field and schema sets) and add their own custom fields and schemas specific to their use cases. By standardizing on a common set of fields, data interoperability is maintained while multiple library operators can compose and share specialized add-on schemas that work in tandem with the core schemas, thus fulfilling their own unique data schema requirements.

Fields are defined on two levels:

1. Validation
    a. Each field must be defined to contain a certain type of data, for example, a string, a number, etc. Other parameters such as length, pattern matching (with regexes) and minimum/maximum values can also be specified.
2. Context
    a. A field's meaning can be clarified by linking it to a definition available at some URL (e.g., <https://schema.org/NGO>).
    b. Other semantic data can also be leveraged to illustrate the relationships between different types of data. For example, how a NGO relates to a more generic organization:
    <https://schema.org/NGO> → is a → <https://schema.org/Organization>
    c. Also, a field can reference a more detailed specification that further defines data inputs and associated labels, thus enabling translation into multiple languages.

Core & add-on schemas

Core schemas are groups of fields that establish a generic actor or object: organization, person, item, service, etc.

Add-on schemas enhance core schemas with further details about the actors' or objects' unique properties. So add-on schemas to the core Organizations schema could include groups of fields describing specific data about a networks' members, or about NGOs, for-profit entities, open source projects, etc.

For example, two networks could both use the Organizations schema, but the first network may pair it with a for-profit company information schema and the other may pair it with a non-profit information schema, each matching their own specific data requirements while ensuring that their core data remains interoperable. The idea is that common fields are

shared using the core schemas, but detailed fields can be added on just by including a second, more specialized schema specific to a specific use case.

Core schemas are fundamental to data interoperability, so they should be used whenever possible to ensure maximum data compatibility. Add-on schemas enable ontological flexibility and can be used to give fuller context to source data.

Indexes

An index is configured to store the data of specific fields that will be searched for by users of the index. For example, an index that records the `country` and `tags` fields enables searches such as "find all nodes in France with the 'bio' or 'agriculture biologique' tag" or "find all nodes in North America with the 'organic' and 'beer' tags".

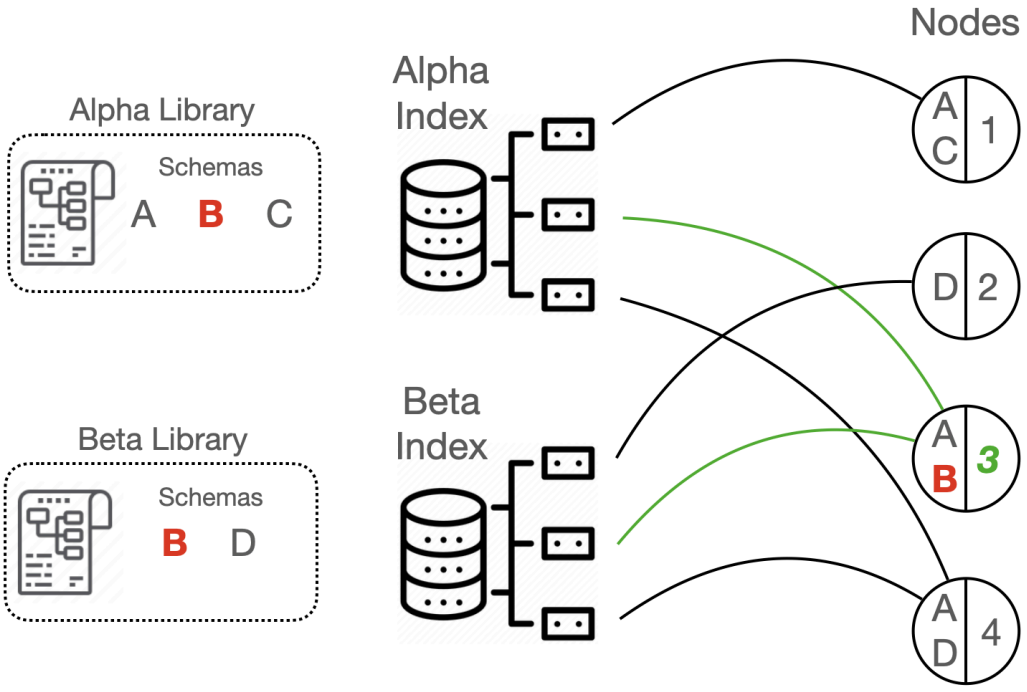
As of April 2024 there are two Murmurations indexes, one for test data and one for real/live data. Recognizing that these indexes centralize one aspect of the Murmurations infrastructure we have now developed detailed deployment documentation to enable other parties to set up their own indexes.

Indexes perform 3 main functions:

- Facilitate node searching
- Track when profiles are created and changed (`last_updated` & `profile_hash`)
- Sync profile changes for schemas shared with peers

Index synchronization

An index can synchronize with other indexes by comparing source data which is common to both indexes for nodes with common schemas. Consider the simple scenario of two indexes (Alpha and Beta) synchronizing.



1. Alpha index and Beta index have the B schema in common in their libraries, and they are peers.
2. Node 3 uses the B schema.
3. Node 3 creates a profile and registers it with Alpha.
4. Node 3 updates its profile, and registers it with Beta.
5. Later during index profile peer syncing, Alpha asks Beta if it has seen Node 3 before.
6. Beta has, and sends the `profile_hash` and `last_updated` timestamp for it.
7. Alpha's `profile_hash` (from step 3) is different from Beta's `profile_hash` (from step 4), so Alpha compares their `last_updated` timestamps.
8. Because Beta's timestamp is more recent than Alpha's, Alpha reaches out to the node to pull the latest version of Node 3's profile and then updates its indexed data and `last_updated` timestamp.

Beta regularly performs the same request to Alpha (and any other peers) in order to update its records relative to Alpha's (and other peer indexes') entries. Tight yet data-efficient synchronization can be achieved if this peer data checking process is nearly continuous (polling every few seconds) between all peers.

Aggregators

Once a network's nodes have been recorded in the index, aggregators can then query the index to find relevant nodes in order to create their own maps and directories.

Because aggregators pull the node data from unknown/untrusted sources, they need to validate source data before importing it into their own databases for further processing. This means ensuring they are receiving valid JSON documents and that the JSON data within them validates to the associated schemas.

To ensure a decentralized network where definitive data about a node is ideally hosted by the node itself, aggregators should always favor data that originates directly from its source. For example, if `xyz.org` hosts its data at its own website (`xyz.org/murm-data.json`), then aggregators should always favor that profile over any other claiming to be about `xyz.org` (e.g., hosted at `many-orgs.net/xyz_org/murm-data.json`).

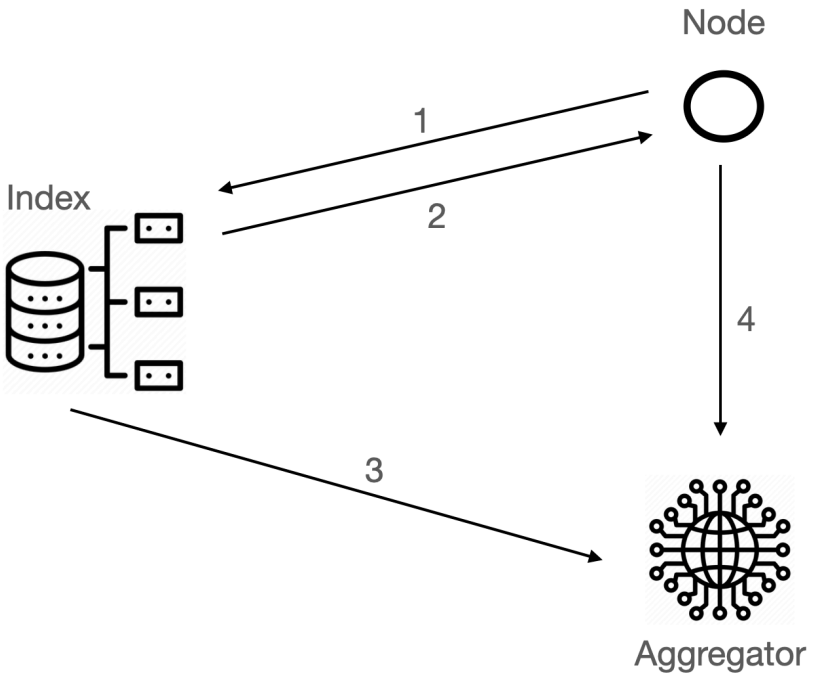
A working example of an aggregator is at <https://wordpress.murmurations.network/software-map/> with its source code here: <https://github.com/MurmurationsNetwork/MurmurationsAggregatorWP>.

Source data & aggregated data

> *Aggregation - the process of combining things or amounts into a single group or total [ref]*

To represent groups of source data in meaningful ways, aggregators collect data from multiple data sources to generate maps, directories, etc. This aggregated data then

represents a snapshot of that grouped source data at the moment the source data was accessed and stored in the aggregator's database. Thus aggregated data is only relevant if it is tightly synchronized with the data source, which is facilitated by checking frequently for updates through an index and then pulling through the latest changes directly from the data source.



1. A node (data source) informs an index about the creation of, or changes to, its profile.
2. The index fetches the profile, validates and then indexes relevant data fields.
3. An aggregator requests the list of nodes from the index that match specific search criteria.
4. The aggregator then fetches all data from relevant nodes for aggregation into maps, etc.

Tools

Murmurations evolved from an idea about defining the DNA of collaboration in 2019 and is now in its second implementation. Version 1, originally built in PHP, has now been completely replaced with a series of open source building blocks written in Golang and ReactJS, which provide various tools to help people use and experiment with the protocol.

We run two identical environments, one for testing and one for production, featuring the following tools:

- An Index Explorer for searching the data in the Murmurations index
- A Profile Generator for creating, and optionally hosting, Murmurations profiles
- A Index Updater for adding, updating and removing profiles from the index

- A Batch importer for registering large batches of CSV data for multiple profiles with the index

We have also built out a range of supporting resources to help you learn about and get involved with the project:

- The Mumurations website provides an introduction and project updates
- Our demonstration maps and directories show example using the components above
- The status reporting tool shows the performance of the network
- Our documentation provide full details explaining how all the components can be used

We have also developed two WordPress Plugins:

1. The Murmurations Profile Generator Plugin - which makes it simple to publish and share details about People, Projects, Organisations, or Offers and Wants from within WordPress
2. The Murmurations Collaborative Map Builder Plugin - which makes it simple to create, curate and update maps and directories of People, Projects, Organisations, or Offers and Wants using WordPress

Future Development

The following concepts should be applied in Murmurations' distributed data sharing protocol to fulfill its goal of facilitating collaboration at scale while enabling individuals and other data creators to maintain control over their data. These concepts would also help ensure authenticity of data and provide measures to prevent spam and other data spoofing and mismanagement scenarios.

Data privacy

At the moment any data shared via Murmurations is open and publicly accessible by anyone. However, Murmuration's data sharing protocol has been designed to enable granular access control over data exchanged over the network by layering well-known and widely used data access and authorization protocols.

For example, indexes could require access control (e.g., authentication protocols such as JWT or API tokens) so that the search results are available only to parties authorized by the index operators.

Nodes may obfuscate their profiles by posting them at randomly named URLs and then posting them only to access protected indexes. More importantly, nodes could selectively encrypt (using public/private key encryption technology) certain data fields so that they can be read only by parties chosen directly by the node. For example, a telephone number could be read only by a select group of individuals, or an email address could only be seen by members of a specific network. Everyone else who has access to the profile will only see an encrypted blob, not the actual information.

Authentication of data (i.e., verifying the data owner/creator) may be managed in a variety of ways. When a node hosts its own data, a certain level of authenticity (or what we like to call "authority") is achieved because the node is hosting statements about itself on its own website. The very fact that the `profile_url` contains the `primary_url` means the index and aggregators have a significant amount of trust that the data is authoritative for that node. In other words, a profile hosted at `xyz.org` is authoritative for the XYZ organization who is associated with that `xyz.org` website address. Exchange XYZ and `xyz.org` with any well-known entity and its website address to fully understand this concept of authority.

Usually third party hosted data implies some sort of authentication of a data source by the third party; one is trusting the third party to vouch for the data source's identity. However, whenever data is signed by the data source, even if it is hosted by a third party it can still be authenticated as coming from its owner. Further authentication of data can be achieved by using the same public/private key encryption technology used for selectively encrypting certain field data. A node can digitally sign field data or even an entire profile, and any index, aggregator or other interested party can verify the data was signed by that node.

Restricted access to and authentication of data (regardless where it is hosted) can be achieved by implementing the selective encryption and data signing on the application layer that interacts with the data stored in profiles hosted by nodes. Although it is not yet implemented, we envision building this functionality into the existing profile generator and also a new data viewing component in our proof of concept data hosting service.

Spam management

There are two basic approaches to managing data spam: allow everything and then selectively deny, or deny everything and then selectively allow. There are different use cases for both approaches, so Murmurations indexes should accommodate either.

The classic approach is to allow everything ("default allow") and selectively deny posting profiles to an index from websites that become known as spam sources. This might become a lot of work for an index operator as it entails constantly updating its deny list and sharing it with peered indexes, as anyone managing an email server can confirm with email spam prevention.

Another approach is to deny everything ("default deny") but then selectively allow certain websites to publish profiles to an index. This approach requires much less ongoing maintenance but its does require index operators and the networks of nodes to confirm a definitive set of websites from which the data will be sourced. The other long-term benefit is that "default deny" enables indexes to significantly simplify filtering and increase accuracy. Additionally, this process of discovering and agreeing members of a trusted network could eventually be automated by leveraging relationships between various nodes and networks as defined in their profiles.

Conclusion

Murmurations directly addresses the aim that has been attributed to the advent of blockchain technology, the first technology that attempted to improve upon the traditional database:

> *What if your database worked like a network — a network that's shared with everybody in the world, where anyone and anything can connect to it? [ref]*

In 2007, futurist and inventor Nova Spivak suggested that Web 2.0 was about collective intelligence, while the new Web 3.0 would be about *connective intelligence*. Unfortunately for the Web 3.0 vision, the vast majority of funding and technical development since Nakamoto's 2008 white paper has centered on the implementation of blockchain. Whilst it is undeniably a powerful technology for managing decentralized transaction data and limiting centralized control, blockchain's "trust-less" relationships and ledgers have not delivered connective intelligence.

By contrast, Murmurations aims to facilitate "trust-based" relationships and transactions which enrich the quality and increase the value of data, whilst making it interoperable and understandable by machines and AI to empower individuals, groups and networks to deliver on the Web 3.0 vision of connective intelligence.

Using the components and concepts described above, Murmurations enables a wide variety of new and intriguing use cases. For example, we envisage Murmurations as a founding metaprotocol for truly decentralized social networking. The next layers we build together on top of Murmurations will drastically increase its utility.

Get started

Our roadmap is at:
<https://github.com/orgs/MurmurationsNetwork/projects/7>

Our current development pipeline is at:
<https://github.com/orgs/MurmurationsNetwork/projects/2>

The source code of the reference implementation of Murmurations can be found at:
<https://github.com/MurmurationsNetwork/>

And documentation is at:
<https://docs.murmurations.network/>

Your can reach out to us at:
<https://murmurations.network/contact/>

Thanks to Matt Slater and Nick Stokoe for their review and feedback on a draft version of this white paper.