Nostr

Nostr - Notes and Other Stuff Transmitted by Relays

The simplest open protocol that is able to create a censorship-resistant global "social" network once and for all.

It doesn't rely on any trusted central server, hence it is resilient; it is based on cryptographic keys and signatures, so it is tamperproof; it does not rely on P2P techniques, and therefore it works.

This is a work in progress. Join the Telegram group!

Very short summary of how it works, if you don't plan to read anything else:

Everybody runs a client. It can be a native client, a web client, etc. To publish something, you write a post, sign it with your key and send it to multiple relays (servers hosted by someone else, or yourself). To get updates from other people, you ask multiple relays if they know anything about these other people. Anyone can run a relay. A relay is very simple and dumb. It does nothing besides accepting posts from some people and forwarding to others. Relays don't have to be trusted. Signatures are verified on the client side.

How to start using Nostr

Nostr client feature comparison

List of projects built on Nostr

This is needed because other solutions are broken:

The problem with Twitter

- Twitter has ads;
- Twitter uses bizarre techniques to keep you addicted;

- Twitter doesn't show an actual historical feed from people you follow;
- Twitter bans people;
- Twitter shadowbans people;
- Twitter has a lot of spam.

The problem with Mastodon and similar programs

- User identities are attached to domain names controlled by third-parties;

- Server owners can ban you, just like Twitter; Server owners can also block other servers;

- Migration between servers is an afterthought and can only be accomplished if servers cooperate. It doesn't work in an adversarial environment (all followers are lost);

- There are no clear incentives to run servers, therefore, they tend to be run by enthusiasts and people who want to have their name attached to a cool domain. Then, users are subject to the despotism of a single person, which is often worse than that of a big company like Twitter, and they can't migrate out;

- Since servers tend to be run amateurishly, they are often abandoned after a while — which is effectively the same as banning everybody;

- It doesn't make sense to have a ton of servers if updates from every server will have to be painfully pushed (and saved!) to a ton of other servers. This point is exacerbated by the fact that servers tend to exist in huge numbers, therefore more data has to be passed to more places more often;

- For the specific example of video sharing, ActivityPub enthusiasts realized it would be completely impossible to transmit video from server to server the way text notes are, so they decided to keep the video hosted only from the single instance where it was posted to, which is similar to the Nostr approach.

The problem with SSB (Secure Scuttlebutt)

- It doesn't have many problems. I think it's great. I was going to use it as a basis for this, but
- its protocol is too complicated because it wasn't thought about being an open protocol at all. It was just written in JavaScript in probably a quick way to solve a specific problem and grew from that, therefore it has weird and unnecessary quirks like signing a JSON string which must strictly follow the rules of [_ECMA-262 6th Edition_](https://www.ecma-international.org/ecma-262/6.0/#sec-json.stringify);

- It insists on having a chain of updates from a single user, which feels unnecessary to me and something that adds bloat and rigidity to the thing — each server/user needs to store all the chain of posts to be sure the new one is valid. Why? (Maybe they have a good reason);

- It is not as simple as Nostr, as it was primarily made for P2P syncing, with "pubs" being an afterthought;

- Still, it may be worth considering using SSB instead of this custom protocol and just adapting it to the client-relay server model, because reusing a standard is always better than trying to get people in a new one.

The problem with other solutions that require everybody to run their own server

- They require everybody to run their own server;
- Sometimes people can still be censored in these because domain names can be censored.

How does Nostr work?

- There are two components: __clients__ and __relays__. Each user runs a client. Anyone can run a relay.

- Every user is identified by a public key. Every post is signed. Every client validates these signatures.

- Clients fetch data from relays of their choice and publish data to other relays of their choice. A relay doesn't talk to another relay, only directly to users.

- For example, to "follow" someone a user just instructs their client to query the relays it knows for posts from that public key.

- On startup, a client queries data from all relays it knows for all users it follows (for example, all updates from the last day), then displays that data to the user chronologically.

- A "post" can contain any kind of structured data, but the most used ones are going to find their way into the standard so all clients and relays can handle them seamlessly.

How does it solve the problems the networks above can't?

Users getting banned and servers being closed

- A relay can block a user from publishing anything there, but that has no effect on them as they can still publish to other relays. Since users are identified by a public key, they don't lose their identities and their follower base when they get banned.

- Instead of requiring users to manually type new relay addresses (although this should also be supported), whenever someone you're following posts a server recommendation, the client should automatically add that to the list of relays it will query.

- If someone is using a relay to publish their data but wants to migrate to another one, they can publish a server recommendation to that previous relay and go;

- If someone gets banned from many relays such that they can't get their server recommendations broadcasted, they may still let some close friends know through other means with which relay they are publishing now. Then, these close friends can publish server recommendations to that new server, and slowly, the old follower base of the banned user will begin finding their posts again from the new relay.

- All of the above is valid too for when a relay ceases its operations.

Censorship-resistance

- Each user can publish their updates to any number of relays.

- A relay can charge a fee (the negotiation of that fee is outside of the protocol for now) from users to publish there, which ensures censorship-resistance (there will always be some Russian server willing to take your money in exchange for serving your posts).

Spam

- If spam is a concern for a relay, it can require payment for publication or some other form of authentication, such as an email address or phone, and associate these internally with a pubkey that then gets to publish to that relay — or other anti-spam techniques, like hashcash or captchas. If a relay is being used as a spam vector, it can easily be unlisted by clients, which can continue to fetch updates from other relays.

Data storage

- For the network to stay healthy, there is no need for hundreds of active relays. In fact, it can work just fine with just a handful, given the fact that new relays can be created and spread through the network easily in case the existing relays start misbehaving. Therefore, the amount of data storage required, in general, is relatively less than Mastodon or similar software.

- Or considering a different outcome: one in which there exist hundreds of niche relays run by amateurs, each relaying updates from a small group of users. The architecture scales just as well: data is sent from users to a single server, and from that server directly to the users who will consume that. It doesn't have to be stored by anyone else. In this situation, it is not a big burden for any single server to process updates from others, and having amateur servers is not a problem.

Video and other heavy content

- It's easy for a relay to reject large content, or to charge for accepting and hosting large content. When information and incentives are clear, it's easy for the market forces to solve the problem.

Techniques to trick the user

- Each client can decide how to best show posts to users, so there is always the option of just consuming what you want in the manner you want — from using an AI to decide the order of the updates you'll see to just reading them in chronological order.

FAQ

This is very simple. Why hasn't anyone done it before?

I don't know, but I imagine it has to do with the fact that people making social networks are either companies wanting to make money or P2P activists who want to make a thing completely without servers. They both fail to see the specific mix of both worlds that Nostr uses.

How do I find people to follow?

First, you must know them and get their public key somehow, either by asking or by seeing it referenced somewhere. Once you're inside a Nostr social network you'll be able to see them interacting with other people and then you can also start following and interacting with these others.

How do I find relays? What happens if I'm not connected to the same relays someone else is?

You won't be able to communicate with that person. But there are hints on events that can be used so that your client software (or you, manually) knows how to connect to the other person's relay and interact with them. There are other ideas on how to solve this too in the future but we can't ever promise perfect reachability, no protocol can.

Can I know how many people are following me?

No, but you can get some estimates if relays cooperate in an extra-protocol way.

What incentive is there for people to run relays?

The question is misleading. It assumes that relays are free dumb pipes that exist such that people can move data around through them. In this case yes, the incentives would not exist. This in fact could be said of DHT nodes in all other p2p network stacks: what incentive is there for people to run DHT nodes?

Nostr enables you to move between server relays or use multiple relays but if these relays are just on AWS or Azure what’s the difference?

There are literally thousands of VPS providers scattered all around the globe today, there is not only AWS or Azure. AWS or Azure are exactly the providers used by single centralized service providers that need a lot of scale, and even then not just these two. For smaller relay servers any VPS will do the job very well.

Protocol specification

See the NIPs and especially NIP-01 for a reasonably-detailed explanation of the protocol spec (hint: it is very short and simple).

Software

There is a list of most software being built using Nostr on https://github.com/aljazceru/awesome-nostr that seemed to be almost complete last time I looked.

NIPs

NIPs stand for Nostr Implementation Possibilities. They exist to document what may be implemented by Nostr-compatible *relay* and *client* software.

- NIP-01: Basic protocol flow description
- NIP-02: Contact List and Petnames
- NIP-03: OpenTimestamps Attestations for Events
- NIP-04: Encrypted Direct Message
- NIP-05: Mapping Nostr keys to DNS-based internet identifiers
- NIP-06: Basic key derivation from mnemonic seed phrase
- NIP-07: `window.nostr` capability for web browsers
- NIP-08: Handling Mentions – `unrecommended`: deprecated in favor of NIP-27
- NIP-09: Event Deletion
- NIP-10: Conventions for clients' use of `e` and `p` tags in text events
- NIP-11: Relay Information Document
- NIP-12: Generic Tag Queries
- NIP-13: Proof of Work
- NIP-14: Subject tag in text events.
- NIP-16: Event Treatment
- NIP-18: Reposts
- NIP-19: bech32-encoded entities
- NIP-20: Command Results
- NIP-21: `nostr:` URL scheme
- NIP-22: Event `created_at` Limits
- NIP-23: Long-form Content

- NIP-25: Reactions
- NIP-26: Delegated Event Signing
- NIP-27: Text Note References
- NIP-28: Public Chat
- NIP-33: Parameterized Replaceable Events
- NIP-36: Sensitive Content
- NIP-39: External Identities in Profiles
- NIP-40: Expiration Timestamp
- NIP-42: Authentication of clients to relays
- NIP-45: Counting results
- NIP-46: Nostr Connect
- NIP-50: Keywords filter
- NIP-51: Lists
- NIP-56: Reporting
- NIP-57: Lightning Zaps
- NIP-58: Badges
- NIP-65: Relay List Metadata
- NIP-78: Application-specific data

Event Kinds

kind	description	NIP
0	Metadata	1
1	Short Text Note	1
2	Recommend Relay	1
3	Contacts	2
4	Encrypted Direct Messages	4
5	Event Deletion	9
6	Reposts	18

7	Reaction	25
8	Badge Award	58
40	Channel Creation	28
41	Channel Metadata	28
42	Channel Message	28
43	Channel Hide Message	28
44	Channel Mute User	28
1984	Reporting	56
9734	Zap Request	57
9735	Zap	57
10000	Mute List	51
10001	Pin List	51
10002	Relay List Metadata	65
22242	Client Authentication	42
24133	Nostr Connect	46
30000	Categorized People List	51

30001	Categorized Bookmark List	51
30008	Profile Badges	58
30009	Badge Definition	58
30023	Long-form Content	23
30078	Application-specific Data	78
1000-9999	Regular Events	16
10000-19999	Replaceable Events	16
20000-29999	Ephemeral Events	16
30000-39999	Parameterized Replaceable Events	33

Message types

Client to Relay

type	description	NIP
EVENT	used to publish events	1
REQ	used to request events and subscribe to new updates	1
CLOSE	used to stop previous subscriptions	1
AUTH	used to send authentication events	42
COUNT	used to request event counts	45

Relay to Client

type	description	NIP
EVENT	used to send events requested to clients	1
NOTICE	used to send human-readable messages to clients	1
EOSE	used to notify clients all stored events have been sent	1
OK	used to notify clients if an EVENT was successful	20

AUTH	used to send authentication challenges	42
COUNT	used to send requested event counts to clients	45

Please update these lists when proposing NIPs introducing new event kinds.

When experimenting with kinds, keep in mind the classification introduced by NIP-16.

Standardized Tags

name	value	other parameters	NIP
e	event id (hex)	relay URL, marker	1, 10
p	pubkey (hex)	relay URL	1
a	coordinates to an event	relay URL	33, 23
r	a reference (URL, etc)		12
t	hashtag		12
g	geohash		12
nonce	random		13
subject	subject		14

d	identifier	33
expiration	unix timestamp (string)	40

Criteria for acceptance of NIPs

1. They should be implemented in at least two clients and one relay -- when applicable.
2. They should make sense.
3. They should be optional and backwards-compatible: care must be taken such that clients and relays that choose to not implement them do not stop working when interacting with the ones that choose to.
4. There should be no more than one way of doing the same thing.
5. Other rules will be made up when necessary.