The Price of Attention: Attaching Bitcoin Fees to Nostr Events

Thomas Voegtlin
electrum.org

November 10, 2025

Abstract

Nostr is the first decentralized social messaging protocol to be successful, thanks to its seamless architecture. However, one remaining issue is spam: How can relays prevent malicious actors from uploading petabytes of undersired generated content? Here we propose a spam filter based on proofs of monetary sacrifice, that can be redeemed by the Bitcoin miners. Users decide how much fee they attach to Nostr events. Relays filter out events that have not received enough fees. Users set a fee threshold under which they do not wish to see content posted by others. This sets a price for spam, and creates a market for attention.

1 Introduction

Nostr is an open and decentralized messaging protocol that gives users control over the content they see. Thanks to its versatility, Nostr has the potential to replace centralized social media applications and platforms, and to free users from algorithmic choices imposed by centralized entities. However, since posting content is essentially free, malicious actors can potentially upload large amounts of data and overload Nostr relays. In the absence of spam filters, users tend to restrict what they see to content posted by accounts they already follow, which tends to amplify the echo chamber effect.

Proof-of-work has been proposed as a spam filter for Nostr [1]. This works by mining event IDs that start with leading zeroes, similar to the hash of a Bitcoin block. One issue with proof-of-work is that various Nostr clients run on different hardware, that are greatly inequal in terms of hashing power. Another proposal, called expensive relay, requires users to pay relays in order to have their events stored and relayed [2]. Unfortunately, there is no way for other relays to verify that the payment was real, and there is no incentive for relays to cooperate.

Here we propose to filter spam by sending fees to the Bitcoin miners, and by attaching public proofs that payments have been made to Nostr events. This is a form of delegated proof-of-work, where purchased hashing power contributes to the security of the Bitcoin network.

2 Notary

Nostr relies on two types of entities: clients and relays. Relays aggregate content uploaded by clients, and relay it to other clients. Users select content they like, and mirror it to other relays. Here we introduce a third type of participant: Notaries.

Notarization is the commitment of data to the blockchain. A notary creates a Merkle tree from an arbitrary large list of documents hashes, and commit the root hash of the tree to the blockchain. The length of each proof is logarithmic in the length of the dataset size. This can be used to create a proof that a document existed at the time the Bitcoin block was mined [3].

Here we use notarization to create a proof that a certain amount was paid for a given Nostr event. Since the spent amount will be claimed by the Bitcoin miners, we call this *proof-of-burn*.

1. Users decide how much fee they want to see attached to Nostr events. Users pay notaries to create proofs-of-burn: a proof that an event ID was commited to the Bitcoin blockchain, and that a certain amount has been paid to the Bitcoin miners. The proof-of-burn is broadcast as a separate Nostr event.

2. Users set a *spam threshold*: An amount of burnt bitcoins under which they do not want to see events posted by other users. This is akin to setting a price on one's attention. Users may relax this rule for pubkeys they follow.

3. Relays set their own spam threshold, under which they will not relay events. Relays may temporarily store events that have not received enough proof of burn. Alternatively, Nostr events and their proof of burn could be relayed together, by implementing a form of package relaying.

4. Users may pay for their own events, or for events posted by others. This implements an upvoting mechanism: It is possible to give increased visibility to someone else's content by attaching fees to it.

Notaries receive micro-payments from users, and users trust that notaries will commit the requested event IDS and burn the requested amount. In order to cover for their expenses, notaries may charge a notarization fee on top of the amount that is burnt. There is no constraint on how many notaries may commit root hashes in the same block (other than the size of the block). A user may pay several notaries simultaneously for the same hash, if they wish to do so.

Notarization is not trustless; a notary could accept payments and keep for themselves the amount they are supposed to burn. This could be mitigated using reputation and public proofs of payment, e.g. the preimage of a Lightning invoice that includes event ID, amount to burn and timestamp in its metadata.

3 Merkle-sum tree

A notary creates a *notarization transaction* that commits the root hash of a Merkle tree to the Bitcoin blockchain, and that sends a given amount to the Bitcoin miners. For each leaf of the tree, we need to create a proof that it contributed to a certain fraction of the total amount burnt by the Bitcoin transaction.

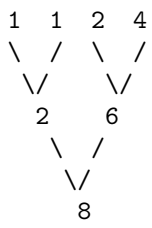

Figure 1: Merkle-Sum tree. The hash of each leaf commits to a certain amount, and the hash of each node commits to the amounts attached to its parents.

We use a Merkle-Sum tree in order distribute the provable burnt amount between the leafs of the tree. Each node of the tree has a hash h and a value v. The hash of an inner node N commits to the hashes and values of its parents [4]:

$$h_N = H(\texttt{Node\_prefix}||h_L||v_L||h_R||v_R)$$

where H is a hash function, L and R denote the node's left and right parents, and `Node_prefix` is a constant. The inner node value is:

$$v_N = v_L + v_R$$

The proofs provided by a Merkle-Sum tree demonstrate that the amount at a given leaf is included in the amount at the root of the tree. The Merkle proof is a list of (h, v) pairs at the neighbour nodes, as well as the position of the leaf in the tree (this is needed in order to know whether we hash with the left or right neighbour).

Merkle-Sum trees are typically used to create proofs of reserves for Bitcoin exchanges. In that case, an exchange can prove to a user that their balance is included in the sum, without the exchange disclosing all account balances. Here the amount at the root of the tree is the amount burnt by the Bitcoin transaction. Thus, proof verification must check that the root hash of the tree is written to the Bitcoin blockchain, and that the burnt amount matches the amount at the root of the tree.

In order to make leaf hashes unique, the user requesting the notarization chooses a random `nonce`, that is included in the leaf hash. The leaf hash is defined as:

$$h_{leaf} = H(\texttt{Leaf\_prefix}||\texttt{event\_id}||\texttt{leaf\_value}||\texttt{upvoter\_pubkey}||\texttt{nonce})$$

where `Leaf_prefix` is a constant, and `upvoter_pubkey` is the public key of the user requesting the notarization.

In order to prove that they requested the notarization, the upvoter creates a signature `upvoter_signature` of h_{leaf}. It is possible to replace `upvoter_pubkey` with blank bytes if one wishes to upvote an event anonymously; in that case, the proof does not require an upvoter signature.

4 Notarization transaction

Paying the Bitcoin miners can be achieved by attaching a large mining fee to the notarization transaction. However, if we did that, miners could add to the blockchain notarization transactions that have never been seen by other miners. This would give them the possibility to notarize content for free.

In order to prevent that, we create a transaction output that can be redeemed by anyone, however with a CSV delay:

```
p2wsh( <csv_delay> OP_CHECKSEQUENCEVERIFY )
```

The CSV delay ensures that the redeeming transaction will be in a different block than the notarization transaction, and thus that the existence of this UTXO will have been public, giving everyone the opportunity to claim it. In practice, Bitcoin miners will be able to claim these funds before everyone else. Therefore, sending to this output is equivalent to a sacrifice to the Bitcoin miners [5][1]. Using a dedicated output for the burn adds the constraint that the burnt amount must be higher or equal to the dust limit.

In addition, the transaction must include the root hash of the Merkle tree. We use an `OP_RETURN` output for that:

```
OP_RETURN( magic_prefix || <root_hash> || <csv_delay> )
```

The `OP_RETURN` output includes a magic prefix and the CSV delay value used in the redeem script, so that Bitcoin miners can recognize the transaction and do not need to guess the CSV delay. Thus, no off-chain information is required to claim the funds. The CSV delay can be chosen by notaries.

A notarization transaction must include a single `OP_RETURN` output and a corresponding burn output. Proof verification must check that the CSV value matches with the scriptpubkey of the burn output.

Note that the existence of distinct outputs for the burn and for the Merkle root hash is dictated by the need to create standard transactions. We use `p2wsh` in the burn output in order to ensure that the transaction is standard. If instead we could use bare output scripts, then it would be possible to write the root hash in the script itself. What matters is that the root hash is visible at the time the notarization transaction is published.

[1] BIP 65 discusses sacrifice to Bitcoin miners using `OP_CLTV` instead of `OP_CSV`. However, for the purpose of retrospectively proving that miners did not cheat, `OP_CLTV` cannot be used. Indeed, a miner could publish the notarization transaction in the same block as the the transaction that redeems it.

5 Proof-of-burn

A proof-of-burn contains the following fields:

1. `chain`: Blockchain genesis hash. Can be omitted if using Bitcoin mainnet.
2. `event_id`: Upvoted Nostr event ID
3. `leaf_value`: Burnt value.
4. `nonce`: Nonce used in `leaf_hash`.
5. `merkle_hashes`: List of `(hash, value)` pairs
6. `merkle_index`: Position in the Merkle tree
7. `block_height`: Block height of the transaction.
8. `txid`: Transaction ID of the notarization transaction.
9. `upvoter_pubkey`: upvoter pubkey (optional)
10. `upvoter_signature`: upvoter signature of h_{leaf} (optional)

In order to verify this proof, a verifier:

1. Verifies that the notarization transaction is in the blockchain (or mempool)
2. Verifies that the notarization transaction has an `OP_RETURN` output, extracts the root hash and CSV delay from it.
3. Verifies that the transaction has a burn output with a scriptpubkey that corresponds to the extracted CSV delay.
4. Reconstructs h_{leaf} from `event_id`, `leaf_value`, `nonce` and optionally `upvoter_pubkey`
5. Verifies that the extracted root hash matches the root hash derived from h_{leaf} and `merkle_hashes, merkle_index`
6. If `upvoter_pubkey` is provided, verifies `upvoter_signature`.

6 Upvoting events

Proofs of burn are published as Nostr events, called *upvoting events*. This does not require any modification of the Nostr protocol, and it is consistent with how Nostr reactions currently work. We use an addressable event ($30000 \leq$ `KIND_UPVOTING_EVENT` < 40000), defined as follows:

```
{
    "kind": KIND_UPVOTING_EVENT,    # addressable
    "created_at": <timestamp>,
    "content": "",
    "tags": [
       ["version", <version>],      # upvoting event version number
       ["chain", chain],            # genesis hash (optional)
       ["e", <event_id>],           # upvoted event ID
       ["d", <leaf_hash>],          # leaf hash
       ["n", <txid>,                # notarization transaction ID
             <block_height,         # block height (0 if unconfirmed)
             <nonce>,               # nonce used in leaf_hash
             <leaf_value>,          # amount burnt
             <merkle_index>,        # position in Merkle tree
             <merkle_hashes>]       # serialized list of (hash:value)
       ["p", <event_pubkey>],       # upvoted event pubkey (optional)
       ["u", <upvoter_pubkey>,      # upvoter pubkey (optional)
             <upvoter_signature>],  # upvoter signature of leaf_hash
    ]
    "pubkey": "<pubkey>",       # pubkey publishing this event
    "id": "<id>",               # id for this event
    "sig": "<signature>"        # signature for this event
}
```

Upvoting events are replaceable, because an unconfirmed notarization transaction may be updated by the notary, which would deprecate all the proofs in that transaction. Event replacement may also be needed in case of blockchain reorgs. Thus, it is convenient to let the notary publish upvoting events; indeed, the upvoter might not be online when the proof needs to be updated.

Note that it is possible to create different upvoting events for the same proof, for example by republishing existing events with a new pubkey[2]. Therefore, in order to correctly count upvotes, one needs to check that they have different leaf hashes. For a given leaf hash, relays should keep only one upvoting event.

Once a proof is final, the upvoter may want to republish it with their own `upvoter_pubkey`. Only then can the `content` field be considered as originating from the upvoter. Until that is done, the content field should be discarded. For a given leaf hash, relays should favor upvoting events that are signed by the upvoter pubkey.

[2]In addition, third parties may replay an already published `upvoter_pubkey` and `upvoter_signature` in a new notarization, which would have the same leaf hash.

7 Conclusion

Proof verification is easier to implement for relays if they run a local Bitcoin node, that can test blockchain inclusion or memory pool acceptance locally. Nostr clients who do not wish to implement blockchain logic may have to trust relays or third parties regarding partial or full proof verification.

In order to avoid waiting for a new block, users and relays may use memory pool acceptance as a good enough proof that fees have been paid. This trusts that the notary will not replace the burn transaction with another transaction that spends to himself. That, combined with a cap on the amount of unconfirmed content seen by relays, might be acceptable in the context of spam protection.

Notaries may replace-by-fee their transactions frequently, in order to add new events to the current Merkle tree. For this to be economical, additional notary fees need to cover the cost of replacing the current transaction. Given enough notarization requests, it is expected that the notarization transaction will undergo multiple replacements, and that its mining fee may overshoot normal memory pool fees. Fee estimators can easily discard those transactions if this becomes an issue.

Relays may want to aggregate the upvotes received by an event, and to relay them together. In addition, if light clients decide to trust relays with proof verification, relays may as well count upvotes received by popular events, and serve the count to clients, instead of sending all the proofs.

If successful, proof-of-burn might create additional income for Bitcoin miners. Proof-of-burn income will be independent from block subsidy, and decorrelated from transaction fees. Interestingly, this proposal creates an incentive for Bitcoin miners to spam unprotected Nostr relays, in order to force users to pay for their posts.

Implementation

A reference implementation of the notary is available at [6].

References

[1] NIP13 `https://nips.nostr.com/13`

[2] Fiatjaf *Expensive Relay* `https://github.com/fiatjaf/expensive-relay`

[3] Peter Todd (2016) *OpenTimestamps: Scalable, Trust-Minimized, Distributed Timestamping with Bitcoin* `https://petertodd.org`

[4] Chalkias et al. (2022) *Broken Proofs of Solvency in Blockchain Custodial Wallets and Exchanges* `https://eprint.iacr.org/2022/043.pdf`

[5] BIP65 *Proving sacrifice to miners' fees* `https://github.com/bitcoin/bips/blob/master/bip-0065.mediawiki`

[6] `https://github.com/spesmilo/notary`