

BrowserCoin: Toward Peer-to-Peer Electronic Cash in the Browser Without Privileged Infrastructure

swompy

www.browsercoin.org

August 2026

Status: this is a design proposal, not a report on finished work. The plural signaling of Sections 3 to 6 is implemented and tested. The local client of Section 8 has been measured but not yet shipped. The manual handshake of Section 9 and the compressed encoding of Section 11 are proposed and not yet built.

Abstract. A purely peer-to-peer currency that runs inside a web browser would let anyone join a monetary network by opening a page, with no installation, no account, and no permission. Existing browser-based implementations do not achieve this. Their consensus rules are decentralized, but every connection between two browsers is brokered by a signaling server the project operates, so whoever controls that server can end the network by switching it off. Decentralization is thereby restored at the level of validation and quietly surrendered at the level of introduction. We show that this dependency is not a property of the browser. A browser cannot accept an unsolicited connection, and roughly eighty-six bytes must reach it through a third party before any direct link can form; but nothing requires that third party to be privileged, persistent, or even aware of the network it is serving. We describe a system in which those bytes travel over several mutually independent carriers at once: the project's own servers, public relays operated by strangers for unrelated purposes, and, in the limit, a human being pasting a short string to another human being. Each carrier suffices alone, none is trusted, and every block is validated locally regardless of which carrier introduced the peer that sent it. The network's survival therefore does not depend on the continued interest, solvency, or cooperation of its author.

1. Introduction

Commerce on the internet has come to rely almost exclusively on software that must be installed, and on identities that must be issued. A currency that runs in a web browser removes both requirements at once: the client is a document, the wallet is a keypair generated on arrival, and participation begins with a URL. This is a meaningful reduction in the cost of joining a monetary network, and it is the premise on which BrowserCoin was built.

The premise contains a trap. Two browsers cannot connect to each other unaided. A page has no listening socket and cannot be dialed; the connection must be arranged in advance by some server that both parties can reach. In practice every browser-based peer-to-peer system solves this the same way, by running one signaling service and pointing all clients at it. The consensus layer is then genuinely trustless. Blocks are validated locally, the longest valid chain wins, and no authority can mint or

censor. But the connectivity layer beneath it has a single owner who can, by inaction alone, prevent any two new participants from ever meeting.

The failure this produces is quiet rather than dramatic. Nothing is stolen and no rule is broken. The existing peers simply keep talking until their tabs close, no new peer is ever introduced, and the network ends by attrition. A system whose continued existence requires one person to keep paying a hosting bill is not decentralized in the sense that matters; it is merely decentralized in the parts that were easy.

What is needed is not the elimination of the third party, which we will show to be impossible inside a browser, but the elimination of the *privileged* third party. If the introduction can be carried by many unrelated networks at once, and if any one of them suffices, then no operator, including the author of the system, occupies a position from which the network can be stopped.

2. The Bootstrap Problem

The constraint is precise and worth stating exactly, because a vague version of it invites hand-waving solutions that do not work.

Browsers establish direct connections using WebRTC, which negotiates a path through NAT using ICE [1]. Before any connectivity check can succeed, the receiving side must possess three values chosen by the initiator: the ICE username fragment, the ICE password, and the fingerprint of the initiator's DTLS certificate. ICE rejects a check whose `USERNAME` attribute does not match the credentials it was configured with, so an unsolicited check from an unknown peer is discarded before any application code observes it. The DTLS fingerprint is the hash of a locally generated certificate and cannot be predicted or negotiated after the fact. These values must therefore arrive out of band.

They are, however, small. The session description a browser produces is a kilobyte or more of text, but nearly all of it is boilerplate that both endpoints already know and could hardcode. Stripped to the values that genuinely vary, an offer is **86 bytes** and an answer **79 bytes**; we verified that a connection opens normally when the full description is reconstructed at the far end from that residue alone.

This reframes the problem usefully. The question is not "how do we run a signaling server" but "how do we move eighty-six bytes, once, from one person to another." Eighty-six bytes fit in a chat message, a QR code, a DNS TXT record, a printed line of text, or a transaction on the chain itself. Almost any channel that carries information at all is a candidate.

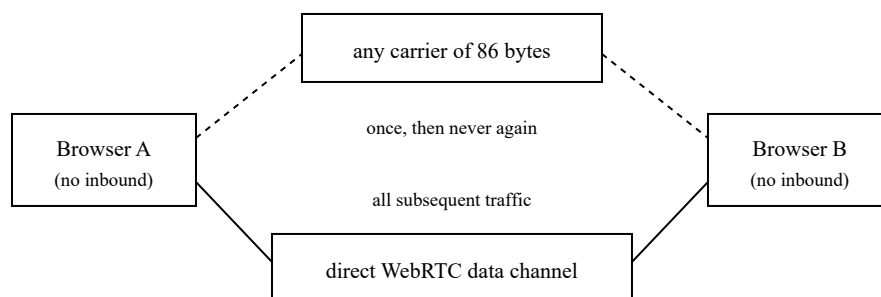


Figure 1: The introduction is carried by a third party; the connection is not.

The asymmetry in Figure 1 is the entire opportunity. The carrier is used once per connection, for a fraction of a second, and handles no chain data even during that interval. Once the data channel opens, the carrier is not merely unused but unreachable. It has no way to observe, interrupt, or degrade a link it introduced. A dependency of that shape is worth very little to whoever holds it, which is precisely why it can safely be handed to strangers.

3. Signaling Substrates

We call any network capable of carrying those bytes a *substrate*. A node runs several concurrently and treats none as authoritative. The interface a substrate must satisfy is deliberately minimal: publish a batch of offers, publish an answer, report whether it is currently up, and accept a flag saying whether we presently wish to hear from anyone. It never touches a peer connection and never sees a block. It moves strings.

Minimality here is not tidiness but reach. Every capability demanded of a substrate is a filter narrowing the set of networks that qualify. We deliberately do *not* require directed messaging, presence lists, persistence, ordering, delivery guarantees, or accounts. What remains is "post a short string where interested parties can see it", and that is satisfied by a very large number of systems that have no idea we exist.

The reference implementation runs two kinds of substrate. The first is the project's own signaling servers, retained because they are fast and already deployed. The second is public relays belonging to Nostr [2], a protocol built for social messaging in which hundreds of independently operated relays accept posts from anyone without registration. Permissionless writing is Nostr's stated design goal rather than an oversight, its wire protocol is four JSON message types over a WebSocket, and its specification already reserves a class of *ephemeral* events which relays forward to current subscribers and are not expected to store. Signaling traffic is exactly that shape, so we are a conforming user of the protocol rather than a parasite on it.

Two properties of this choice matter more than the protocol details. First, WebSocket connections are not subject to the same-origin policy, so a page can reach any relay from any origin, including a page loaded from the local filesystem, which we return to in Section 8. Second, and decisively, the relays exist for reasons unconnected to this project and will continue to exist whether or not anyone here remains interested. That is the difference between infrastructure one depends on and infrastructure one merely uses.

4. The Swarm

Because a substrate is not required to support addressed messages, dialing cannot be modelled as calling a known peer. It is modelled instead as a swarm, in the manner long used by peer exchange in file distribution [3]:

1. A node with spare capacity creates several connection offers, each backed by its own peer connection, and publishes them addressed to nobody in particular.
2. A node seeking peers observes an offer, produces an answer, and publishes it addressed at that one offer.
3. The two connect directly. The substrate is finished.

Offers and answers are published only after ICE gathering has completed, rather than trickling candidates as they are discovered. This costs a second or two of setup latency, which is immaterial for a gossip mesh that has no interactive deadline, and reduces substrate traffic by roughly an order of magnitude by turning a stream of small messages into exactly two. When the carrier is somebody else's unpaid infrastructure, being economical is not an optimization but a condition of being welcome.

The offer pool is small and finite. The reference implementation keeps four. Publishing a single offer would render a node invisible from the moment anyone accepted it until its next republication; publishing many would leave a popular node holding dozens of half-open connections it must maintain and then discard. Every pending offer and every partial handshake carries an expiry, and a periodic sweep closes whatever has aged out and refills the pool, so no combination of lost messages, vanished peers, or misbehaving relays can leave the dialer permanently stuck.

Identity across substrates is a single node identifier shared by every path. A peer discovered simultaneously through a project server and through three relays is recognised as one peer and adopted once. Without this, redundancy in the introduction layer would translate directly into duplicate connections.

5. Load Proportional to Demand

A design that borrows other people's infrastructure must bound what it takes, or it will be excluded by the operators it depends on, and deservedly so. The mechanism is a single lever: a node that has reached its peer target goes silent *and deaf*. It stops publishing offers, and it also unsubscribes, so it no longer receives anyone else's.

Deafness is the half that is easy to omit and expensive to omit. If well-connected nodes continued to listen merely because listening feels free, then every node would receive every other node's announcements, and inbound traffic per node would grow linearly with the size of the network. A network of ten thousand would impose ten thousand times the load of a network of ten on each participant and on each relay. This is the classic broadcast failure, and a reliable way to be banned from public infrastructure.

With the lever in place, both published and received traffic are proportional to the number of nodes actively seeking peers, not to the number of nodes that exist. In a mature network the seeking population is a churn quantity: nodes arriving, nodes that lost a peer, nodes below their minimum. It does not scale with the total. Section 11 gives the arithmetic, including the point at which this alone stops being sufficient.

6. The Direct Link

Replacing a signaling library with raw peer connections removes two services that were previously invisible, and the second of them is a correctness issue rather than a convenience.

The first is encoding. Application messages are structures; a data channel carries strings and binary. Serialisation on the way out and parsing on the way in is unremarkable.

The second is fragmentation. The transport beneath a data channel will not carry an arbitrarily large message: 16 KiB is the largest size every implementation reliably accepts, and an oversized send may fail outright or, worse, tear down the channel. A BrowserCoin block may reach 256 KiB and is hex-encoded on the wire, so a single block message approaches half a megabyte and a range response is larger still. A mesh link without fragmentation therefore appears to work. Handshakes succeed, small messages flow, status indicators turn green. It then fails precisely when a joining node attempts to synchronise history, which is the moment it matters most.

Messages are consequently framed and chunked explicitly, with an eight-byte header per fragment:

```
u32 message id | u16 fragment count | u16 fragment index | payload
```

Reassembly is keyed by message id rather than assuming ordered arrival, since a peer may interleave fragments of concurrent messages. A ceiling on reassembled size and a cap on concurrent partial messages prevent a peer from pinning memory by opening many messages and completing none. Sending applies backpressure at the channel's outbound buffer, because pushing thirty-two fragments in a tight loop can balloon the buffer and, in some implementations, close the channel.

None of this is novel. It is recorded because it is the part that a description of the architecture would omit and an implementation cannot.

7. Validation as the Only Trust Anchor

Distributing introductions across strangers' infrastructure is safe for one reason, and it is worth being explicit about what that reason does and does not cover.

A substrate cannot cause a node to accept an invalid block. Every block that arrives is verified locally against the consensus rules: proof of work, signatures, balances, difficulty. The path by which its sender was introduced is not an input to that decision. A hostile relay that forwards fabricated chain data achieves nothing, because it is not forwarding chain data at all; the most it carries is an invitation, and an invitation from an attacker leads to a connection with an attacker, who is then subject to the same validation as everyone else.

What a hostile substrate *can* do is decline to introduce us to honest peers, or introduce us only to peers of its choosing. This is an eclipse attack, and it is bounded here in the same way Bitcoin bounds the equivalent risk in its DNS seeds [4]: run several carriers of different provenance simultaneously, retain other discovery paths, and remember that a node need only reach the honest network by *one* route to be corrected. The attacker's task is not to control a relay but to control every relay a victim uses, along with every other path, at the same time.

Peer addresses learned over existing links matter here too. Once a node holds even one honest connection, it learns of further peers directly from the mesh, needing no substrate at all. Substrates are the way in, not the way to stay.

8. The Client as a File

If the network no longer requires a server to introduce peers, it is worth asking what remains of the requirement to host the client. We measured this rather than assumed it. A page loaded from the local filesystem is, in current Chromium and Firefox, a complete runtime: a secure context with WebCrypto, local storage, an indexed database, workers, WebRTC, outbound WebSocket connections, and unrestricted cross-origin fetch. Every capability the node uses is present.

We confirmed the consequence end to end. Two separate browser processes, each with its page loaded from the filesystem, formed a direct data channel through a public relay with no project-operated infrastructure of any kind involved. The client therefore need not be served. It can be a single file of approximately six hundred kilobytes that a user saves, copies to a memory stick, sends as an attachment, or hosts anywhere, and which functions identically wherever it lands.

This closes a gap that plural signaling alone leaves open. A network that survives its author losing interest, but whose client must be downloaded from that author's domain, has merely moved the single point of failure from the signaling host to the web host. A client that is a file removes both. The domain becomes a convenience for newcomers rather than a dependency of participants.

9. Human Signaling

The final substrate is not a network. Section 2 established that an introduction is 86 bytes; a person can carry 86 bytes. No part of what follows is implemented today; it is described as a design because it is the rung on which the whole argument finally rests.

One node produces a short string and its operator sends it to another person by any means whatsoever: a chat message, an email, a photograph of a screen, a spoken sequence of characters. The recipient pastes it into their node, receives a reply string, and returns it. The two browsers then connect directly, with no server involved beyond the address-discovery step discussed in Section 11. The exchange is a round trip and requires both people to be present, and the strings expire after a few minutes, so it is a live handshake rather than a posted invitation.

Its significance is out of proportion to its convenience, for two reasons. The first is that it terminates the regress: below this rung there is no infrastructure left to disable, because the carrier is a human being choosing to help another human being. Any system that can suppress it can suppress conversation.

The second is that it need happen only once. A node that completes a single manual handshake is inside the mesh, and thereafter learns of other peers from the peers it already has. The manual path is a door, not a commute. The cost is paid at entry and never again. A community cut off from every relay simultaneously can rebuild its connectivity from one person pasting one line into a group chat.

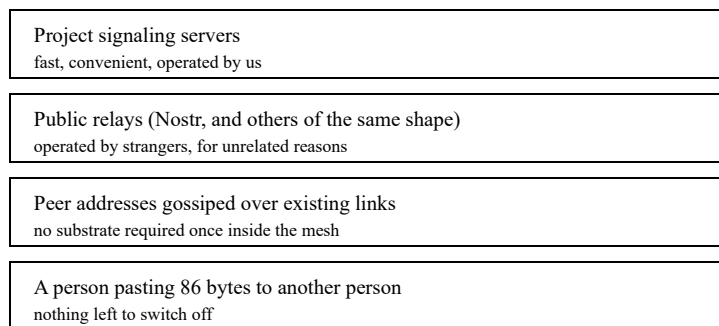


Figure 2: Degradation ladder. Each rung is slower than the one above and harder to remove.
Any single rung is sufficient to join the network.

10. Privacy

An honest account must record what this design exposes, since a session description contains network addresses.

Publishing offers to public relays would place participants' addresses in a plaintext stream that anyone may read and that archivists do read. Payloads are therefore sealed with a key derived from the network's name. This is obfuscation and not confidentiality: the key is derivable by anyone who reads the source, and it is described as such rather than presented as encryption. Its purpose is narrow and real: to deny bulk collectors a ready-made map of every participant's address, while conceding that a determined observer who specifically wants the node list can have it. Against the prior arrangement it is not a regression in any case, since a signaling server necessarily saw the same addresses in the clear.

The manual path of Section 9 deserves a blunter warning, and the interface should give it: a string pasted into a public channel publishes the sender's network address to everyone who can read that channel. Between acquaintances this is unremarkable. In a public forum it is a disclosure the user must make deliberately, and software that allows it silently is not being neutral.

Address discovery is discussed as a residual dependency in Section 11 rather than here, but its privacy character is worth noting: the servers involved are stateless, learn only that some browser asked its own address, and never carry chain data or observe peers.

11. Calculations

Availability. Let a node run k substrates, each independently available with probability p . Signaling fails only if all fail together:

$$P(\text{no introduction possible}) = (1 - p)^k$$

With six relays at a pessimistic $p = 0.9$, this is 10^{-6} . The figure is however less informative than it appears, because the assumption of independence is doing all the work. Six relays behind one hosting provider are one substrate wearing six names, and the exponent collapses. The defensible engineering claim is therefore about provenance, not count: relays are selected across distinct operators, hosts and

jurisdictions, and a diverse list of four is worth more than a homogeneous list of twenty. Correlated failure, not individual reliability, is what the parameter must be chosen against.

Substrate load. Let N be network size, f the fraction of nodes actively seeking peers, R the relays each node uses, T the republication interval, and B the size of a published announcement. A seeking node uploads RB/T and receives approximately fNB/T ; by Section 5, a satisfied node does neither. Taking $T = 45$ s, $R = 6$, and $f = 0.05$:

Announcement B	Upload per seeker	$N = 10^3$	$N = 10^4$
Full descriptions, ~11 KB	1.5 KB/s	12 KB/s	122 KB/s
Compressed, ~0.5 KB (proposed)	0.07 KB/s	0.6 KB/s	5.6 KB/s

The upload column is negligible in both rows; the inbound column is where the design lives or dies. Publishing complete session descriptions, as the reference implementation currently does, is comfortable into the low thousands of nodes and becomes antisocial beyond that. 122 KB/s of inbound traffic is a burden on the participant and an imposition on relays that agreed to none of this. The 86-byte reconstruction of Section 2 is therefore not a micro-optimisation but the difference between a design that scales past ten thousand participants and one that does not.

Beyond that, the remaining lever is to stop having every seeker hear every other seeker. Partitioning the rendezvous point into m topics by node identifier divides inbound traffic by m at the cost of introductions confined to a shard, which is acceptable for a gossip mesh that requires only a connected graph and not a complete one. Load then grows as fN/m , and m may itself be adjusted as the network grows.

Connectivity. Direct connection fails for the pairs where both endpoints sit behind address-dependent mapping, conventionally estimated at 10–15% of pairs. Conventional practice is to add relay servers that carry the traffic itself; we decline, because such a server would both cost money and sit in the middle of real chain traffic, reintroducing exactly the privileged position this design exists to remove. The cost is tolerable because a gossip mesh does not require every pair to connect, only enough pairs for the graph to remain connected. With a target of eight peers, a per-pair failure rate of 0.15 leaves the expected degree near seven, far above the threshold at which a random graph of this size partitions. The affected minority retains the manual path of Section 9.

Residual dependency. One remains. To tell a peer its address, a node must know it, and a browser behind NAT knows only a private address. Address discovery uses public STUN servers, which are stateless, carry no traffic, observe no chain data, and are operated by many unrelated organisations; the browser races several and uses whichever answers first. We looked for a way to remove even this, by having connected peers report the address they observe, and found it closed: implementations redact the remote address in connection statistics, so the mesh cannot learn its own reflexive address from itself. We record the negative result rather than omit it. The honest claim is not that the system requires no servers, but that it grants none of them a privileged position. That is the same claim Bitcoin can make about its seeds, and no weaker.

12. Conclusion

We have described a browser-based currency in which no infrastructure operated by the project is load-bearing. The introduction of peers, previously the single point at which one party could end the network by inaction, is carried by several mutually independent substrates: the project's own servers, public relays that exist for unrelated reasons and are operated by strangers, addresses gossiped between peers already connected, and finally a person passing a short string to another person. Any one suffices. None is trusted, because a substrate can withhold an introduction but can never cause a valid node to accept an invalid block. The client itself need not be hosted, being a single file that runs from a local filesystem with full capability.

The residual dependency is a stateless address-discovery service, replaceable by any of many unrelated providers, and the residual limit is that some fraction of pairs cannot connect directly and must fall back to a human introduction. Neither confers control on anyone.

What is proposed here is therefore not a network with no servers, which the browser forbids, but a network with no privileged ones. It continues because participants find it useful and help each other join, rather than because its author continues to pay for it. That is a weaker guarantee than the phrase "fully decentralized" is usually made to carry, and a considerably stronger one than most systems making the claim can support.

References

1. A. Keranen, C. Holmberg, J. Rosenberg, "Interactive Connectivity Establishment (ICE): A Protocol for Network Address Translator (NAT) Traversal," RFC 8445, 2018.
2. Nostr, "Notes and Other Stuff Transmitted by Relays," protocol specification, github.com/nostr-protocol/nips.
3. A. Norberg, L. Strigeus, G. Hazel, "Peer Exchange," BitTorrent Enhancement Proposal 11, 2008.
4. S. Nakamoto, "Bitcoin: A Peer-to-Peer Electronic Cash System," 2008.
5. E. Heilman, A. Kendler, A. Zohar, S. Goldberg, "Eclipse Attacks on Bitcoin's Peer-to-Peer Network," USENIX Security, 2015.
6. J. Rosenberg, R. Mahy, P. Matthews, D. Wing, "Session Traversal Utilities for NAT (STUN)," RFC 5389, 2008.