

The bstack: publish once, prove everything, and let users ride free

Most of a modern application stack exists to do two jobs the application itself never asked for: copying data to everyone who needs it, and getting strangers to trust what arrives. This paper presents the pattern language that removes both jobs, the basis of a family of applications we call the bstack. State is carried by Bitcoin transactions that prove themselves; a multicast network delivers each publication once to every subscriber; hosts are interchangeable replicas competing on service; readers verify instead of trusting. The patterns themselves run over plain unicast too, peer to peer to the hosts; multicast adds reliable distribution with global reach. It sets out the open standards the patterns compose, the transaction shapes, the delivery model, and the economics: participation free at the edge, delivery metered like a commodity in the middle, revenue at every tier that serves an audience. It closes with what still costs money, and the limits stated plainly.

Author: Jeff Harris · Lightweb Inc. · jeff@lightweb.net

Paper 009 · **Version** 03

Audience: application architects, CTOs, and technical evaluators of Bitcoin infrastructure.

We = Lightweb Inc. and the 1BSV network; **you** = an architect deciding what your next system rides on. No background in Bitcoin or in multicast networking is assumed; each term is explained where it first appears. This architecture is in rapid development: the design and this paper will change with it, and the version stamp beside the author line identifies this revision.

The short version

- **One pattern language yields a whole application tier.** A directory, an audit log, a message box, an organization and a secrets manager are the same handful of patterns wearing different rules.
- **The foundations are ratified open standards.** Every layer is a published BRC, the Bitcoin SV ecosystem's numbered public specifications: self-proving objects, overlay state, key derivation, payments, discovery. The design act is composition; the one new layer is delivery.
- **The transaction is the data structure.** Mine the state, not the data: a few hundred bytes reach the chain per update (327 to 440 B measured on the worked example), while the data itself rides transactions built never to be mined, so a retraction is a real retraction.
- **Commodity multicast deletes the distribution tier.** Multicast means the network itself copies one transmission to every subscriber. One publication reaches every subscribed host at the same moment, in place of host-count squared crossings, and the fan-out servers, webhook farms, polling loops and reconciliation jobs go with it.
- **The patterns do not wait for the network.** The same objects travel over plain unicast, peer to peer to each overlay host, with nothing strictly needed but a good block-header source; multicast adds reliable distribution with global reach, at a cost flat in the audience.
- **Users ride free, and paid content sits above the floor.** Publishing costs miner fees and nothing else, and the base answer is free and anonymous on every conforming host, by specification: a host that charged for it would have to know who was asking. Richer questions, history, proofs and keyed content are all priceable above that floor.
- **Receivers pay commodity rates, and the structure survives every hop.** Each host pays for the volume it elects, flat in the number of participants; each tier monetizes its own audience, and what arrives by broadcast can be re-broadcast downstream.
- **Hosts compete on service, never on data.** Every host holds the same objects because the network delivered them; availability, latency, retention and richer questions are the market, and a wrong answer is caught by verification, not by reputation.

1. One shape, many applications

Ask a modern application where its engineering budget went and the answer is rarely the feature list. It went into moving state between services and regions, keeping copies consistent, and building the apparatus of trust: accounts, sessions, audit systems, and the operational promise that the database was not quietly edited.

The bstack is a family of applications built so that neither job exists. Each application publishes its state as Bitcoin transactions, and a multicast network delivers each publication once to every host that subscribed to the application's topic. Two properties of a Bitcoin transaction do the heavy lifting, and they are the only Bitcoin background this paper assumes. A transaction is signed, so nobody can alter it after the fact. And it can carry a short proof tying it to the block chain, so anyone who holds the chain's headers (80-byte block summaries; the full history of them fits comfortably on a phone) can check it without asking any server. Checking a transaction against headers alone is called SPV, simplified payment verification, and it appears throughout this paper.

The name is the pattern family, not a product. The worked example throughout is bfinger, a Bitcoin-era descendant of the old UNIX `finger` utility that shares nothing with it but the question: ask what a name currently claims about itself, and get back an answer that proves itself. bfinger is the first member of the family to ship, not a template. Later members stand on their own, drawing on the same pattern language and adding patterns of their own where their problems demand them.

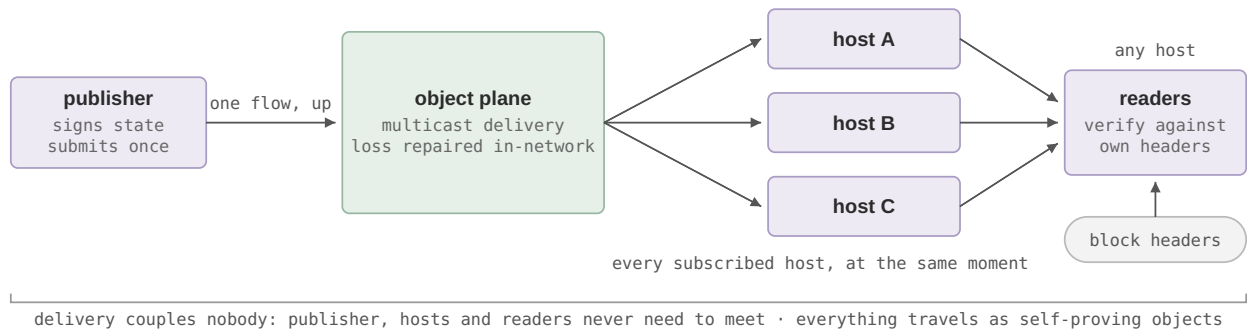


Figure 1 - the shape every bstack application shares: one directional flow up from the user, broadcast out to every subscribed host, and readers who verify rather than trust.

Four roles, and the pairs you would expect to be coupled never need to touch. The publisher signs state and submits it once. The network delivers it to every subscribed host. A host indexes what it received and answers questions. A reader asks any host, or several, and checks everything it is handed against block headers whose proof of work it checks itself, from a source of its own choosing. Publishing never involves a host directly, reading never involves the publisher, and no host talks to another. Everything between them travels as objects that carry their own evidence, which is why a host can be added, swapped or lost without anyone reconfiguring anything.

To be precise about the scope of that separation: it is a property of publishing and delivery, not a wall between the parties. A reader can always approach a publisher directly, to request exclusive access to published content, for instance, and the keyed-content and payment patterns later in this paper exist for exactly that conversation. The architecture's claim is narrower and stronger: nothing in getting data published, delivered, served or verified ever depends on that contact happening.

bfinger is deliberately modest, and its unglamorous surface is the point: every pattern in this paper is visible in a command with four lines of output, and every one of them carries unchanged into far larger designs.

2. The foundations are ratified standards

A BRC is a numbered, public specification in the Bitcoin SV ecosystem's open standards repository; citing one is citing a published contract, not a private design. Nothing in section 1 required a new consensus rule, a proprietary protocol, or a token. The patterns compose standards that are already ratified, and the composition is the design act.

Layer	Open standards	What the patterns take from it
Self-proving objects	BRC-62, 74, 95, 96	a transaction travels with the proofs that tie it and its ancestry to block headers, so anyone holding the headers verifies it with no third party
Overlay state	BRC-22, 24, 88	topics admit outputs under application rules; lookup services answer questions over the result
Keys and identity	BRC-42, 43, 100	one identity key derives every child key it needs: signing, funding, payments; wallets speak one interface
Discovery	BRC-169, 180	<code>user@domain</code> resolves to a key, and a domain names the services that answer for it
Script devices	BRC-48, 60	data-bearing outputs locked to derived keys; transactions that can be held off the chain indefinitely
Payments	BRC-29	any identity key is already a payable destination, fresh output per payment, nothing to register
Restriction	BRC-369	where confidentiality is required: keyed content and conditional key release, never pricing
Markets	BRC-178	competing hosts race to answer; lookup data priced as a commodity
Distribution	BRC-148, 149	the object plane: publish once, and every subscribed host of the topic receives it

Everything above the last row predates this work; the delta the bstack adds is a delivery substrate and a record discipline. Overlay networks made Bitcoin applications scalable, and BEEF (the envelope format of BRC-62, which packs a transaction together with its proofs) made the overlay object self-proving. The companion paper *The overlay object plane* presents the delivery layer itself; this paper is about what becomes buildable once it exists.

3. Transaction composure: mine the state, not the data

Two Bitcoin mechanics carry this whole section. A transaction spends outputs of earlier transactions, and the network's consensus guarantees each output is spent at most once: competing spends are resolved by miners, and everyone converges on the same winner. That guarantee, double spend protection, is the only lock this architecture ever uses. The chain does the one thing only a chain can do, putting events in an order nobody can rewrite; everything else stays off it, where storage is cheap, copies are free, and a retraction is real. Six devices, composed per application.

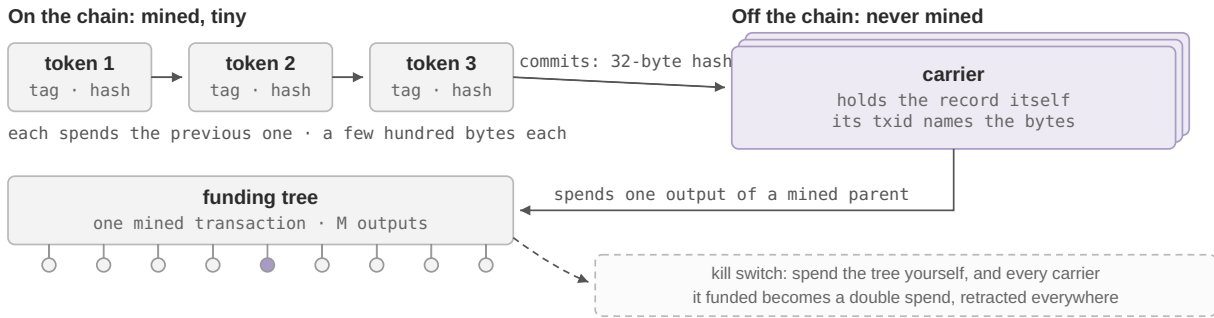


Figure 2 - mine the state, not the data: the chain carries order and commitment in a few hundred bytes, while the data rides carriers that can always be proven and always be retracted.

The state token. A chain of tiny mined outputs, each spending the previous one, each committing to the current state. Spends are the one thing consensus already serializes, so versioning, rotation and retirement inherit double-spend protection instead of reimplementing it. The history cannot fork, and every host holds the same token at the same moment.

The carrier: a transaction built never to be mined. The record itself rides an ordinary transaction whose fee is zero and whose lock time is a century away, so no miner can include it and none would want to. The wrapper is free, and it pays for itself three times over: the carrier's transaction id is a 32-byte name for exactly those bytes; the carrier spends an output of a parent that is mined, so it is provable by ordinary SPV even though it will never be in a block; and it can be revoked, as the next device shows.

The funding tree: postage and admission in one. One mined transaction with M small outputs; each published object spends exactly one. Every object is one proven parent plus one short proof, constant size, independently verifiable, and structurally rate-limited: no unspent output, no publication. Flooding costs real money at the door, with no rate-limiter service to run.

The kill switch. Spend the funding outputs yourself and every carrier hanging off them becomes a double spend at once; conforming hosts drop the records. One mined transaction retracts a record, or everything a tree ever funded. Deletion is usually a request; here it is a spend. Only the key holder can make it: a carrier's signature has exactly one valid encoding, and hosts count only a spend that can be mined, so a copy of a carrier re-encoded by anyone else retracts nothing.

The interval anchor. For high-rate streams, no object is mined at all: one anchor transaction per interval commits a single 32-byte digest (a Merkle root) over the interval's objects. Chain time and a completeness proof for the whole interval, at one miner fee per interval regardless of rate.

Sub-store roots and the witness. A record commits to entire stores of further carriers by the same kind of 32-byte root, so one mined output validates data amalgamated across many stores, revealed a member at a time or all at once. And each record publishes the hash of a secret it withholds, revealing the secret only in the next transition: holding the identity key is not enough to advance the state, you must also hold what the previous state committed to.

What reaches the chain, measured on the worked example's published test vector:

Operation	Reaches the chain	Never reaches the chain
create	one token, 327 B	the record, inside its carrier
update	one token, 440 B	the new record
retract	one sweep, 398 B	every carrier that funding paid for

The chain carries the order and the commitment; the data lives where a retraction is real.

4. Distribution: what commodity multicast removes

4.1 The problem it replaces

Overlay applications already had self-proving objects. What they did not have was distribution. An object reaches interested hosts one HTTPS submission at a time: the publisher performs the fan-out, work grows with hosts times topics, success means one acknowledgement, and a host that missed an object gets no signal that it did. Between N hosts propagating to each other, one object crosses the network on the order of N squared times, fully verified at every arrival before a duplicate check discards it. Two hosts of the same topic can diverge silently. Everyone who is not a host learns state by polling.

None of that is weak engineering: it is plain HTTPS, it needs nothing beyond the hosts themselves, and at a handful of hosts the costs are invisible. But it prices audience growth into every publisher and every host, and it makes "did everyone hear?" unanswerable.

4.2 Publish once

Multicast deserves one plain sentence before it does any work. In unicast, a sender opens one connection per recipient and transmits the same bytes again for each of them; in multicast, the sender transmits once and the network replicates the stream at its branch points, so the sender's cost does not grow with the audience.

The object plane applies that mode to application objects. A publisher submits an object once, named by topic; the network delivers it to every host subscribed to that topic, complete, verbatim, proofs intact. Loss is detected and repaired inside the network itself; repair scales with loss experienced, not with audience

size. What arrives is the same bytes the reader will verify, so the network never has to be trusted, only used.

A topic is just a name. On the wire it becomes the SHA-256 hash of that name: a fixed 32-byte identifier drawn from a namespace of 2^{256} values, so applications mint topics freely, with no registry, no approval step, and no realistic chance of two names colliding. Subscribers elect the topics they want and receive nothing else. A namespace that large, riding a fabric that must physically organize who hears what, is also an open frontier: how topical subscription topologies are laid out, automated and re-optimized as interest shifts is one of the research areas this architecture opens, and one of the ones we find most exciting.

Subscribed hosts	Host-by-host propagation	On the plane
10	on the order of 100 crossings	1 submission
100	on the order of 10,000	1
1,000	on the order of 1,000,000	1

Audience size stops being a cost the publisher engineers against and becomes a property the network provides.

4.3 The two savings

Development. An application on the plane never builds the distribution tier: no fan-out servers, no webhook farms, no polling loops, no reconciliation jobs between its own services, no replication protocol between hosts, and no origin infrastructure whose failure takes the data offline. A host is an engine, its rules, and a subscription. The consistency machinery that normally consumes the roadmap is replaced by a transport property: every subscribed host heard every object.

Physical resources. The same subtraction lands on the wire. Host-count squared crossings become one submission and in-network replication; per-host verification of duplicate deliveries becomes one verification of one arrival; polling becomes silence until there is something to say. The companion papers price this in settlement form: unicast distribution concentrates the whole fan-out on whoever serves the data, and fronting it with a CDN relocates the bill without removing the amplification, at a cost that scales with participants times traffic. An architecture whose cost grows with every participant prices participation out; emit-once distribution is flat where it matters.

4.4 One flow, one direction

Notice what the shape in Figure 1 does not contain: request loops. Data moves in one direction, starting at the user. A publisher pushes one flow up through the nearest door; the network fans it out; hosts index what arrived; readers pull only at the last step, and only questions, not firehoses. There is no origin to poll, no cache hierarchy to invalidate, no backhaul of the same bytes over the same links per consumer. The efficiency is structural rather than clever: each link carries a given object a bounded number of times, however many parties end up serving it.

4.5 Unicast first; multicast for reach

Nothing in the pattern language waits for the plane. The same self-proving objects travel over ordinary unicast connections, straight from a publisher to each overlay host it cares about, peer to peer, and every pattern in this paper holds: the committed records, the funding trees, the free base answer, the interchangeable hosts, the payments. The only infrastructure a participant strictly needs is a good source of block headers to verify against.

That posture is fine at a handful of hosts, and it is where an application can start today. What it reintroduces is section 4.1: the publisher performs the fan-out, misses are silent, and audience growth is priced into every participant. Multicast adds exactly what unicast cannot: reliable distribution, with loss detected and repaired by the network itself, and global reach at a cost flat in the audience. **The patterns are the architecture; multicast is how the architecture scales.**

5. Interactions, and what each party learns

The privacy properties fall out of the same shape, and each comes with its limit stated.

What the chain shows. One mined output per update: a derived public key (a child key computed one-way from the identity key, so an observer cannot connect it back), a four-byte application tag, and a 32-byte salted commitment. Not the name, not the domain, not a field of the record, not even its length. The record itself is never written into a block, which is why retracting one actually retracts it. The limit: anyone who already knows your identity key can derive the same child key and read the timing of your updates; this is pseudonymity against a stranger, not concealment from an acquaintance. Two levers would tighten the posture further: a fresh derivation per update removes even that linkage from the chain, at the cost of finding a record from its key alone, and unlinkable one-time addressing breaks the visible thread entirely, moving discovery wholly onto the hosts. Both are named levers, not defaults.

What a read reveals. On the base answer, nothing identifies the reader: no account, no credential, no session, no payment. The host learns which subject was asked about and from what address; the chain learns nothing, because a lookup touches nothing mined. Free reads are anonymous reads, and that is a privacy property before it is a pricing one: a host that charged for the base answer would have to identify the payer, and every read would become attributable. Paid questions and paid content exist above that floor, and a paid read is attributable by nature, which is exactly why the anonymous base answer is pinned beneath it rather than left to host policy.

What a payment reveals. The simplest payment derives a fresh destination from the recipient's verified identity key, producing an ordinary output only the two parties can recognize, settled point to point with nothing touching a topic or a host. Richer payment shapes are part of the design space rather than exceptions to it: a payment can travel inside a record and settle when its recipient claims it, split across many receivers, be posted as a bounty that competing providers race to claim, or run as an interactive flow that an overlay's own rules settle. What holds across all of them is that the delivery meter never bills a payment as a payment, and the payment graph is only as public as the design chooses to make it.

What everyone can see. A record on a public topic is a bulletin board entry: public in full to every host, and written accordingly. Where less must be disclosed, a record commits to stores by root and reveals members selectively; where nothing must be disclosed, content is encrypted under keyed-content release, and

hosts hold ciphertext they cannot distinguish from noise. Metadata (sizes, timing, who addressed whom on a shared topic) remains visible to hosts; designs that need less visibility say so rather than pretending otherwise.

6. Economics: free at the edge, commodity in the middle

6.1 Who pays for what

Flow	Who pays	Basis
Publishing state	the publisher	miner fees on a few hundred mined bytes per update; the data itself is never mined
Submitting to the network	nobody	admission is free; it is protocol work, not a metered product
Delivery to a host	each receiving host	delivered volume only, flat in the number of participants; repair is never billed
Reading the base answer	nobody	free on every conforming host, by specification
Richer questions and keyed content	the reader	host-priced classes, exports and proofs; a payment that releases a key; the free base answer is untouched
A payment	the payer	point to point in the simplest case, plus one miner fee; richer shapes in 6.3

The meter sits on the party receiving value, and nowhere else. A user publishes and reads without an account existing anywhere; the paying parties are the operators who elect to receive flows because those flows feed a business.

6.2 Why the free edge is load-bearing

Free participation here is not a promotion, it is an invariant with mechanics behind it. A price attaches to a question class, never to part of an answer: the base classes are priced zero on every conforming host, a priced class may not be a superset of a free one, and a host must refuse a question carrying a member it does not define, or any extra word would manufacture a priceable alias of a free question. The base answer's bytes are identical on a host that charges for nothing and one that charges for everything else, which is what lets readers compare replicas at all.

The deeper distinction: a restriction is a property of bytes, and over a broadcast plane only encryption produces one; a service fee is a property of a socket, binding one reader to one host for one answer. A priced lookup therefore sells availability, latency, retention and convenience, never exclusivity, and must never be presented as access control. Systems that confuse the two sell their users a lock that is not there. Paid content that must stay paid is keyed content: the payment releases the key, and the restriction lives in the bytes themselves.

The floor is a floor, not a ceiling. Everything above the free base answer is open to pricing: richer question classes, history beyond a window, exports, availability proofs, and keyed content sold per release. Free participation is what keeps the system readable and anonymous at its base; the paid tiers are where hosts and publishers build businesses.

6.3 Revenue at every tier, structure preserved

The delivery meter never bills a payment as a payment, so where does revenue live? At every edge that faces an audience.

- **Hosts monetize their readers.** Richer question classes, history beyond a window, exports, retention guarantees, per-request payments on a 402 (HTTP's own "payment required" status), subscriptions: all of it host-side, none of it changing what the free base answer returns. A host's revenue is from its readers, not from the publisher or from the network.
- **Publishers pay for reach the way they pay for postage.** Miner fees and structural postage make spam expensive while leaving publication cheap; the incentive to place data on the network is the value of the data itself.
- **The network sells delivery as a commodity.** Metered on elected volume, flat in participant count, with admission and repair never billed. Every operator's bill tracks its own appetite, not the network's growth.
- **Distribution stacks downstream.** What arrives by broadcast can be re-broadcast: an operator that receives a flow once can extend delivery across its own multicast domain to its own audience, on the same open standards, becoming a distributor in its own right. Each tier pays commodity rates for what it elects and monetizes the edge it owns, so the economic structure of the network survives every hop outward.
- **Payment topologies stay open.** The point-to-point leg is the simplest shape, not the only one: a bounty published as a record that providers race to claim, a series of bounties funding ongoing service, a payment split across many receivers, an interactive flow settled by an overlay's own rules with every host judging it alike. The delivery meter bills none of them as payments, so new payment shapes never disturb the commodity economics underneath.

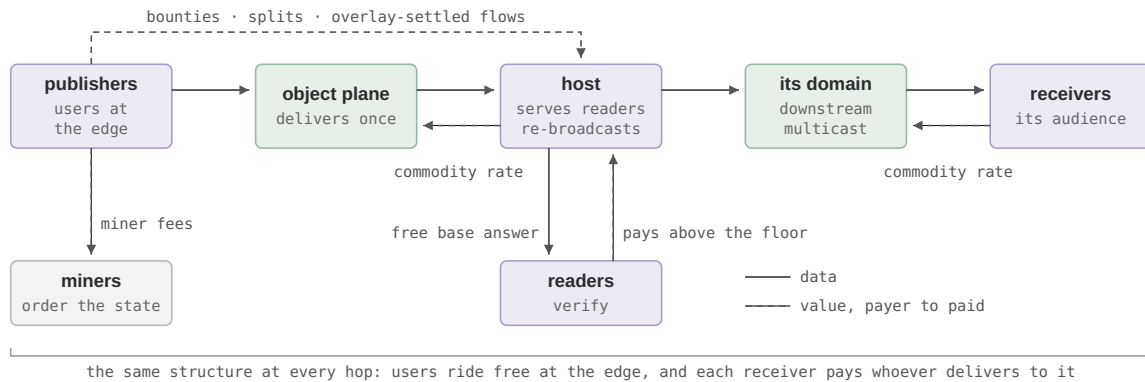


Figure 3 - money follows the structure: users publish and read the base answer free, every receiver pays a commodity rate to whoever delivers to it, and the same shape repeats at every hop outward.

The tiers compound rather than contend: every host the plane adds is one more subscriber every publisher reaches at no added submit cost.

7. Hosts compete on service, never on data

In a conventional architecture the party holding the data holds the application. Here the network hands every subscribed host the same objects, each host admits them on their own validity, and nothing one host knows could not have been received off the same plane. Three consequences follow.

Choosing a host is an availability question, not a trust question. A host that answers wrongly, whether stale, forked or malicious, is caught by verification; the cost of picking a bad host is a refusal and a retry, never a wrong result. Readers who care ask several hosts and require agreement, and a disagreement is surfaced, not averaged away.

Entry is a subscription. Adding a host is joining a topic; the publisher never learns it exists and no existing host is reconfigured. The replica set grows the way an audience grows, not the way a cluster grows.

The market is in the service, because the data is a commodity. Hosts holding identical bytes compete on availability, latency, retention depth, coverage, and the richer questions they choose to price. Race-settled markets take this to its conclusion: hosts compete per query and the reader pays the one it chooses, with collective availability, not exclusive custody, as the product. No moat forms around the data itself, and that is by design: the moat was always where the users are.

8. Assumptions and limits

Quantity	Value	Basis
Mined bytes per operation	327 B create, 440 B update, 398 B retract	measured: the worked example's published test vector
Miner fee policy	100 satoshis per kilobyte	stated: prevailing BSV mainnet policy
Chain cost per update	about 44 satoshis at 440 B	derived: mined bytes times fee policy
Funding tree width M	16 outputs per mined tree	stated: the worked example's default
Host-by-host crossings	on the order of N squared for N hosts	structural: each host re-propagates to every other advertising host (Paper 008)
Plane submissions per publication	1	structural: BRC-148/149 delivery
What a mined output discloses	a derived key, a 4-byte tag, a 32-byte salted hash	verified: the committed-record specification of the worked example

Limits, stated plainly:

- The conventional stack is faster to the first feature, and that is worth conceding in full.** A managed database, a CDN and a SaaS directory need no new concepts and hire from a deep pool. What they cannot give at any spend is the property this design starts from: answers a reader verifies without trusting the operator, and distribution whose cost does not grow with the audience.
- Metadata on a shared topic is public.** Content can be encrypted and payments are invisible to the plane, but sizes, timing and addressing among writers are visible to every host. Designs that need less visibility must say so and pay for it, not imply it.
- Retraction is not erasure, and it binds conforming hosts, not physics.** The kill switch removes future availability from honest hosts; it cannot un-copy what somebody already fetched, and nothing can. The sweep itself is a small self-proving transaction, so any reader who holds it can prove the retraction to anyone; proving that no sweep exists is the hard direction, because header verification alone cannot show an output unspent. Bounding how long a dishonest host can serve a retracted record (validity windows, spend-check services, signed answers that turn a violation into portable proof) is named open work.

4. **A price cannot restrict.** Every subscribed host holds a copy, so confidentiality comes from encryption and key release, never from a paywall on a lookup. Any design claiming otherwise is selling a lock that is not there.
5. **"The same moment" is an architecture property, not physics.** Flows are numbered and order is recoverable per flow, and no ordering across subscribers is promised; applications that need total order put it in the data, with a chain, which is exactly what the state-token pattern is for.
6. **A chain reorganization re-anchors proofs; it does not fork state.** Occasionally the chain's newest few blocks are replaced and their transactions are mined again into different blocks. Every party here verifies against headers whose proof of work it checks, so acceptance follows the strongest chain automatically: a proof that pointed into a replaced block fails until its refreshed proof is republished and delivered like any other object. Ordering rests on spends, which miners arbitrate, so the state chain does not fork; an application that must be safe against deep reorganization waits for confirmation depth, exactly as any Bitcoin payee does.
7. **Scale assumes the plane; the patterns do not.** Every pattern here runs over ordinary unicast connections, publisher to hosts, with nothing strictly needed but a trustworthy block-header source; section 4.5 states what that posture gives up. Distribution at reach rides a commercially operated multicast network, and history bootstrap and catch-up remain the overlay ecosystem's pull synchronization either way. The companion papers state what the network side costs and where its meter sits.

References

Authoritative BRC specifications at <https://github.com/bsv-blockchain/BRCs>:

- [BRC-22](#) Overlay Network Data Synchronization
- [BRC-24](#) Overlay Network Lookup Services
- [BRC-29](#) Simple Authenticated BSV P2PKH Payment Protocol
- [BRC-42](#) BSV Key Derivation Scheme (BKDS)
- [BRC-43](#) Security Levels, Protocol IDs, Key IDs and Counterparties
- [BRC-48](#) Pay to Push Drop
- [BRC-60](#) Simplifying State Machine Event Chains in Bitcoin
- [BRC-62](#) Background Evaluation Extended Format (BEEF) Transactions
- [BRC-74](#) BSV Unified Merkle Path (BUMP) Format
- [BRC-88](#) Overlay Services Synchronization Architecture
- [BRC-95](#) Atomic BEEF Transactions
- [BRC-96](#) BEEF V2, Txid Only Extension
- [BRC-100](#) Wallet-to-Application Interface
- [BRC-148](#) Multicast Shard Domain Partitioning and the BEEF Object Plane
- [BRC-149](#) Multicast BEEF Object Frame Format
- [BRC-169](#) Universal Handle Addressing and Resolution for the Metanet
- [BRC-178](#) Race-Settled Collection Markets for Overlay Lookups and Message Boxes
- [BRC-180](#) Overlay Service Discovery at an Internet Domain
- [BRC-369](#) Keyed Content and Conditional Key Release

Companion papers:

- *The overlay object plane: publish BEEF once, and every overlay hears* (Lightweb Inc., Paper 006): <https://1bsv.net/papers/overlay-object-plane.pdf>
- *The overlay bridge: an unmodified engine on the object plane* (Lightweb Inc., Paper 008): <https://1bsv.net/papers/overlay-bridge.pdf>
- *The Asset service at scale* (Lightweb Inc., Paper 002): <https://1bsv.net/papers/asset-pull-and-the-cdn-reflex.pdf>
- *The cost of being reachable* (Lightweb Inc., Paper 003): <https://1bsv.net/papers/ingress-economics.pdf>
- *Arcade v2 on a multicast delivery fabric* (Lightweb Inc., Paper 007): <https://1bsv.net/papers/arcade-on-the-fabric.pdf>

1BSV - BSV Layered Multicast - cash moves fast on 1BSV. · 1bsv.net

Lightweb Inc. · Jeff Harris · jeff@lightweb.net · github.com/lightwebinc

Unbounded Solutions for a Small World™ · © Lightweb Inc.