

The overlay object plane: publish BEEF once, and every overlay hears

Overlay networks made Bitcoin applications scalable, and BEEF made the overlay object self-proving: a transaction travels with exactly the evidence needed to verify it. Neither solved distribution. Today an object reaches the hosts that want it one HTTPS submission at a time, paid for by the publisher, with no signal when a host misses one. This paper presents the missing layer: the BEEF object plane (BRC-148/149), a dedicated global multicast plane that delivers each published object to every subscribed host of its topic at the same moment, complete and verifiable, with loss repaired by the network itself. It recaps the overlay and BEEF background, explains the multicast solution, shows what an application architecture stops carrying, sketches the applications global multicast unlocks, and closes with how the 1BSV network operates the plane commercially.

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

Paper 006 · **Version** 05

Audience: overlay application developers, wallet and service engineers, and the architects who own their infrastructure.

We = the 1BSV fabric; **you** = an overlay application developer.

The short version

- **Overlay propagation is still point-to-point.** Under BRC-88 a publisher discovers the hosts interested in a topic and submits the object to each of them over HTTPS: work grows with hosts × topics, success means at least one acknowledgement, and a host that misses an object gets no signal that it did.
- **BEEF made the object self-proving; the object plane makes it self-delivering.** BRC-148/149 give overlay objects a dedicated multicast plane of their own: publish once, and every subscribed host of the topic receives the object verbatim, proofs intact.
- **Topics are the delivery primitive.** Any topic name works immediately, there is no registry to apply to, and a subscriber elects exactly the topics and encodings it wants: nothing else is delivered.
- **Reliable delivery is a network feature, not your code's job.** Loss is detected and repaired inside the fabric itself; what arrives is complete, and subscribed hosts never silently diverge.
- **Your architecture sheds its distribution tier.** No fan-out servers, no webhook farms, no polling loops, no reconciliation jobs between your own services: publish once, and everything you run hears together.
- **Verification stays sovereign.** Each object arrives with its Merkle proofs and is checked against your own header feed: SPV certainty at every subscriber, and for every end user behind one, with no trust chain between services.
- **A released overlay engine joins through one small bridge.** Delivery arrives as the submit call the engine already serves, and publication goes through the bridge once, in place of the engine's own host-by-host propagation; the companion paper *The overlay bridge* gives the setup.
- **1BSV operates the plane, in early access.** Publishing an object through the public door is free, like sending a transaction; receiving is the paid service, billed on delivered volume only.

1. Overlays, and the step that never scaled

1.1 What an overlay is

An overlay network is a state machine riding Bitcoin. A **topic** names a set of on-chain outputs and the rules for membership; a **topic manager** (BRC-22) decides which outputs of a submitted transaction are admitted into the topic and which spent coins leave it; **lookup services** (BRC-24) answer queries over the resulting state. Topics are permissionless and conventionally named `tm_*` (BRC-87). A token ledger, a payment rail, a file-commitment registry, and a game's move history are all just topics with different managers.

The interchange format of this world is already BEEF. Current overlay implementations accept a submission as BEEF bytes plus a list of topic names, verify the transaction by SPV, apply each named topic's rules, and return per-topic admittance instructions. The model is sound, deployed, and genuinely permissionless.

1.2 How objects move between hosts today

Distribution is where the architecture shows its age. Under BRC-88, a publisher (or a host propagating onward) discovers the hosts interested in a topic via SHIP advertisements, then delivers the object to each of them individually over HTTPS (Figure 1).

If this were weak engineering it would not have carried the overlay ecosystem this far. It is plain HTTPS, it needs no infrastructure beyond the hosts themselves, discovery through SHIP and SLAP is elegant, and at a handful of hosts per topic the costs are invisible. The default success criterion is honest about the economics: a broadcast counts as delivered when **at least one** responsive host acknowledges each topic. The rest of the interested set is reached eventually, by relay, or not.

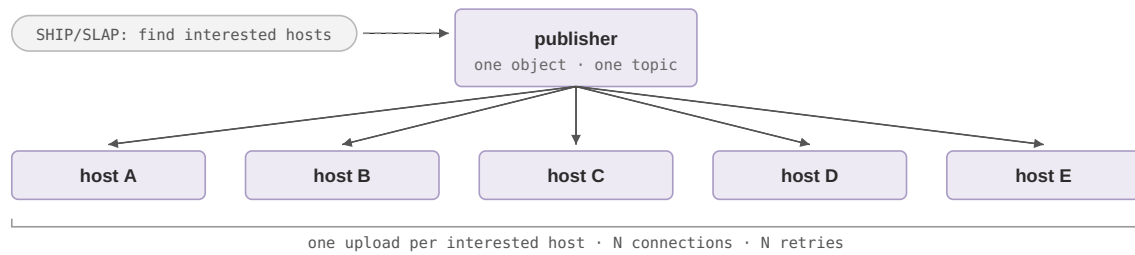


Figure 1 - per-host propagation under BRC-88: the publisher performs the fan-out itself, once per interested host, and pays for every copy.

1.3 The three structural costs

- **The publisher performs the fan-out.** Every interested host is a separate upload, connection, acknowledgement, and retry policy. Work scales with hosts × topics, and it lands on the party the ecosystem wants more of: the publisher.
- **Delivery is unobservable.** HTTP tells a publisher a request succeeded; it cannot tell a host what it never received. There is no numbering across objects on a topic, so a missed object is not an error anywhere: two hosts of the same topic can diverge silently and discover it only by comparison, after the fact.
- **Everyone else polls.** Systems that are not overlay hosts themselves (the app backends, indexers, and analytics pipelines behind them) learn state by querying lookup services on a timer. Polling adds latency where freshness matters and load everywhere else.

What request/response legitimately buys is discovery, lookup, and history: questions a client asks once and wants answered from state. Those stay. The step that does not belong on request/response is steady-state propagation: the same object, wanted by many parties, at the same time. That step is a broadcast, and the network should carry it as one.

2. BEEF: the object that proves itself

BEEF (Background Evaluation Extended Format, BRC-62) packages a transaction with exactly the evidence a receiver needs to verify it independently: the BUMP Merkle paths (BRC-74) anchoring its ancestry to mined blocks, and any unmined ancestors in between, arranged so validation can proceed as the bytes stream in. The receiver checks the Merkle roots against its own block headers, evaluates the scripts, and needs no indexer, no UTXO set, and no third party's word. The specification's worked example is 677 bytes: proof-carrying transactions are small.

Three encodings make up the family, each self-identifying by its leading bytes:

Encoding	Leading marker	Purpose
BEEF	0100BEEF	the baseline: BUMPs + ancestry + subject
Atomic BEEF	01010101 + subject TXID	one explicit subject transaction (BRC-95)
BEEF V2	0200BEEF	TXID-only references to known data (BRC-96)

BEEF is what overlay hosts already accept, what BRC-100 wallets already emit, and what SPV was always supposed to feel like. But BEEF is a **format**, not a **transport**: it says nothing about how an object reaches the many parties who want it. Today that gap is filled by the per-host uploads of Figure 1. The rest of this paper fills it properly.

3. The object plane: multicast built for overlay objects

3.1 The fabric underneath, in four properties

BSV Layered Multicast is the open standard for moving Bitcoin data as structured broadcast instead of gossip or per-peer unicast. The fabric BRCs (BRC-124, BRC-126, BRC-129, BRC-130) treat the mechanics in depth; an application developer needs to hold onto only four properties:

- **Emit once.** A sender's cost is one emission regardless of audience. The network replicates at branch points; no link carries a second copy.
- **Enter at the nearest door.** Ingress is anycast: a publisher reaches the closest entry point, and the audience's location is never the publisher's problem.
- **Numbered flows.** Every stream carries per-flow sequence numbers, so absence is observable: a receiver knows what it did not get.
- **Self-repair.** Anything lost in transit is detected and redelivered by the network itself; repair scales with loss experienced, not with audience size.

The network scales horizontally toward its design target of 1B+ transactions per second, and none of that machinery is visible at your integration surface.

3.2 A plane of its own

Two words carry the rest of this paper. A **lane** is one independently numbered stream on the fabric that a receiver can elect: the raw transaction stream is a lane, the block headers are a lane, and each lane arrives separately, so a receiver takes exactly the lanes it asked for and nothing else. A **plane** is a group of lanes with its own publishers, subscribers, and addressing, operated and scaled as a unit. The fabric's original plane is the **settlement plane**: the lanes miners use to build blocks (transactions, subtrees, blocks).

BRC-148 gives overlay traffic a delivery plane of its own, the **object plane**, running alongside the settlement plane and never mixed with it: different payload, different publishers, different subscribers, operated and scaled independently. A subscriber takes two lanes: the BEEF object lane, which carries the objects,

and the block-header lane delivered beside it, which carries the reference those objects are verified against. The plane carries BEEF objects **verbatim**. The fabric never parses, translates, or re-encodes them, and no proof data is ever stripped: the bytes a publisher submits are the bytes every subscriber receives.

3.3 Subscribe by topic

On the object plane, the topic is the unit of addressing, subscription, and delivery. A topic is identified by the hash of its name (its TopicID: SHA-256 of the UTF-8 topic name), and a developer can lean on three consequences of that:

- **No registration.** Any topic name works, immediately. The topic space stays as permissionless as BRC-22 left it, and there is no registry to apply to, wait on, or pay for.
- **Election is precise.** A subscriber's election is the pair (*topics*, *encodings*): you receive the topics you elected, in the encodings you can process, and nothing else. A host that cannot resolve TXID-only ancestors simply excludes BEEF V2.
- **Objects arrive whole.** Large objects are split and reassembled by the network's standard machinery, with loss repair underneath. Your code sees complete BEEF objects, never fragments.

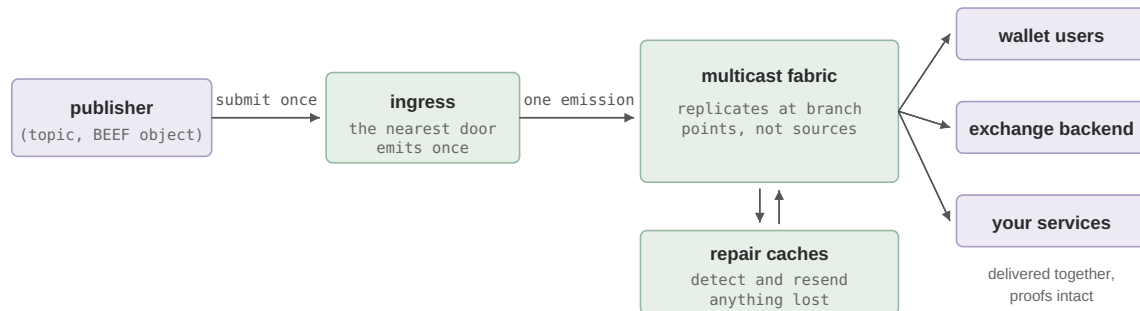


Figure 2 - the object plane: one submission, one emission, every subscribed host of the topic served together, with repair standing by.

3.4 Delivery you can audit

Every stream on the plane is numbered end to end, gaps are detected and repaired automatically, and ordering is recoverable per flow: the same delivery machinery that carries the settlement plane, applied to overlay applications. And every mechanism your integration touches is a published open standard (BRC-148/149, and the fabric suite behind them); nothing in the delivery path asks for trust.

4. Building on the plane

The plane has three integration surfaces: the submission record and the delivery record, both specified in BRC-149, and the block-header lane beside them, carried per BRC-135. All three are small. If you run a released overlay engine rather than your own receive loop, you need not touch any of them: the companion paper *The overlay bridge* shows one small bridge landing all three on the engine as it ships, unmodified.

4.1 Publishing

A publisher submits one record: the topic name (or names) and the BEEF object, verbatim. That is the entire publish-side contract; the ingress takes it from there and emits the object once per named topic.

- **Single-topic submission is an open class.** It has the same bounded footprint as sending a transaction, so it goes through the public door with no account and no fee, like any send.
- **Open does not mean unbounded.** Every ingress door bounds what a single source may publish through it, and what the door as a whole may put onto the object plane; every delivery edge bounds what one subscriber is sent, which is where a flood spread across many doors converges. An object refused at either point is never delivered and never metered, so an anonymous flood cannot reach a subscriber's socket and cannot reach its bill.
- **A submission names up to 15 topics, and every name travels.** One record carries the object once, whatever it is named to, and every subscriber that receives it sees the whole list. How many of the leading names are delivered on depends on the door: the public door delivers the first topic, an authenticated connection the first three. The rest are labels, carried but never routed, so an anonymous record can never become fan-out and a paying subscriber is never sent, or charged for, an object it did not elect. A subscriber that elected several of the delivered topics receives the object once.
- **Proof updates are first-class.** Identity on the plane is the object's bytes, never its subject TXID, so a payment published at send time and the same payment republished with its refreshed BUMP once mined are two legitimate, distinct objects. Subscribers see the payment instantly and its mined proof moments after the block: a pattern webhooks never made cheap.

4.2 Subscribing

A subscriber elects topics and encodings, and its delivery lane streams one length-delimited record per delivered object:

Offset	Size	Field	Meaning
0	32	TopicID	the matched topic's identifier
32	4	ObjectLen	object byte length (uint32 BE)
36	...	Object	the BEEF object, verbatim, proofs intact

1. The delivery record per BRC-149: 36 bytes of framing per object; the subscriber maps TopicIDs back to the names it elected.

Beside the object lane runs the block-header lane: the 80-byte headers, delivered as the chain advances, which are the reference BEEF verification needs. The receive loop is the BRC-62 validation procedure, unchanged: parse the object, compute Merkle roots from its BUMPs, match them against your header store, evaluate scripts, admit per your topic manager's rules.

4.3 What stays where it is

The plane replaces steady-state propagation and only that. Discovery (SHIP/SLAP), lookup (BRC-24), transaction history (BRC-64), and catch-up (BRC-76 Graph Aware Sync, and the block-anchored completeness check of BRC-136) all remain the province of the existing overlay protocols: a host that was offline backfills through them, then rides the stream. Your topic managers keep deciding admittance; the plane just ensures that every host running one receives the same objects at the same time.

5. What your architecture stops carrying

Consider what an overlay application at any real scale builds today: fan-out workers and webhook endpoints for partners, retry queues and backoff policies per destination, pollers for every system that cannot accept a push, and reconciliation jobs to detect the divergence all of the above eventually produces. None of it is the product. All of it exists because distribution was left to the application.

Axis	Per-host propagation (today)	The object plane
Publisher work	one upload per interested host	one upload, any audience
Who fans out	the publisher, then relaying hosts	the network, at branch points
Adding a subscriber	publishers must learn of and add it	it joins the topic; no publisher work
Simultaneity	staggered, publisher-paced	every subscriber served together
A missed object	invisible; found later by comparison	detected and repaired by the network
Proof carriage	intact (BEEF over HTTPS)	intact (BEEF verbatim on the plane)
Freshness for backends	polling lookup services	pushed, at arrival
Discovery & history	SHIP/SLAP, lookup services	unchanged: SHIP/SLAP, lookup services

The plane turns your audience size from a cost you engineer against into a property the network provides.

Three effects reach further than the table:

- **Your own services stop drifting.** Regions, replicas, and pipelines all subscribe to the same topics and hear the same objects at the same moment, each verifying independently. There is no primary to fan out from, so there is no fan-out to fall behind.
- **Bandwidth tracks interest, not the chain.** You receive the topics your users care about, never the settlement firehose. A quiet topic costs nearly nothing to watch.
- **Certainty compounds downstream.** Because every object carries its proofs, each subscriber can hand SPV-grade certainty to its own users without adding a trust hop. The chain of "our API says so" becomes a chain of Merkle paths.

6. What global multicast unlocks

Some of what follows is today's work done without today's plumbing. The more interesting part is what becomes reasonable to build when a single publish reaches every subscribed party on the planet at once, proof attached:

- **Planetary token and payment rails.** An issuer publishes each transfer to its topic once; every exchange, custodian, market maker, and wallet user receives it in the same instant and verifies it independently. Partner integration collapses from N webhook contracts to one sentence: subscribe to the topic.
- **Market data without privileged copies.** Venue events and on-chain trades reach every participant together, off one emission, each copy verifiable against the subscriber's own headers. Nobody sits downstream of an API gateway, so nobody is structurally last.
- **Machine-scale feeds.** Sensor fleets, oracles, and telemetry anchored on-chain publish for free through the public door; subscribers elect only the topics they need. A million devices publishing is not a million connections to anything.
- **Shared-state applications.** Games and collaborative apps treat the chain as their shared state and the plane as their state bus: a move published to the app's topic lands on every regional service simultaneously. Sessions and chat stay on ordinary connections; state rides broadcast, and the bespoke sync tier disappears.
- **Consortium workflows with built-in audit.** Supply-chain and inter-company processes subscribe every member to shared topics: the same signed artifact lands everywhere at once, provable then and provable later, with no central hub to operate or to subpoena.

- **Commitment registries.** UHRP-style file commitments and similar registry entries are advertised once to the registry topic; every host hears immediately, while resolution stays on lookup services where it belongs.
- **Distributed storage.** Data that must exist in several places at once (archives, replicas, vaults across regions and providers) is published once and lands everywhere at the same moment, each copy carrying the proof that makes it verifiable; a replication tier becomes a subscription.
- **Indexes and directories.** State that many parties look up (registries, directories, catalogues) is a token its owner updates; every index hears each change together, so any host answers the same question the same way, with proof, and no single host has to be trusted.
- **Data distribution.** Streams of application data (segments, batches, events) ride the plane as objects: one publish per item, every subscriber served at once, cost tracking interest rather than audience.

The common shape: **publish-side simplicity, subscribe-side precision, and verification at every subscriber.** When those are network properties, application architecture gets smaller as the audience gets larger.

7. 1BSV: the operated plane

BSV Layered Multicast is the open standard; **1BSV** is the commercial network Lightweb operates on it. The object plane is live on 1BSV, in early access for application accounts. What the operated network adds to the specifications is reach and operations: anycast ingress doors, the global fabric between them, repair caches standing by everywhere, and delivery edges placed near you.

7.1 The account model

1BSV serves three account tiers, picked at signup, and the tier decides which lanes an account may elect. The mapping is strict: a lane outside an account's tier is neither delivered nor billed.

- **Miner accounts** receive the settlement plane: the transaction stream and the subtree and block lanes, sized for block production, and they alone may publish blocks and subtrees into the network.
- **Application accounts**, the tier this paper is written for, receive the object plane: **BEEF topics plus the block-header lane**, and never the raw transaction stream, whose volume is sized for mining operations.
- **Service-provider accounts** sit between the two: a transaction broadcaster with a proof service (an Arcade v2 stack, for instance), a wallet backend, or an SPV service mines nothing and holds no settlement privileges, but builds on settlement data. Such an account receives the transaction stream, subtree data, and the block-header lane. Block data for a broadcaster's proof service is arranged with us rather than elected from the standard menu.

7.2 Integration, in five steps

1. **Open an application account.** Self-service signup with one approval step, from the account console.
2. **Generate your keys.** Your tunnel is standard WireGuard; you create the keypair and we only ever receive the public half.
3. **Elect your delivery.** Request a tunnel, elect the BEEF object lane and the block-header lane beside it, and say where in your network delivery should land. Approval provisions exactly that.
4. **Connect.** Bring the tunnel up. Every tunnel is provisioned with a primary and a standby path with automatic failover.
5. **Subscribe, verify, serve.** Choose the topics and encodings you want, and change them whenever you like: subscriptions are yours to manage after approval, from the account console. Objects start arriving as the length-delimited records of §4.2; run the BRC-62 loop against the header lane and hand your users certainty.

Running a released overlay engine? Step 5 is the bridge's job, and *The overlay bridge* covers it.

There is no proprietary SDK and no private wire format: **the published BRCs are the whole client contract.** Everything your code touches (submission records, delivery records, frames, BEEF itself) is an open specification you can implement, test, and audit independently.

7.3 Where the meter sits

- **Publishing through the public door is free.** It needs no account, like any send. A record may name up to 15 topics on any door; the public door delivers the first, an authenticated connection the first three, and the rest travel as labels. There is no charge for naming topics on either door.
- **Receiving is the paid service.** Billing is volume-only: you pay for the overlay objects and proofs delivered to you, and for nothing else. No subscription fees, no minimums, no commitments.
- **Never billed:** traffic you send into the network (measured, shown to you, free), and the network's own repair retransmissions.
- **Miner fees are separate and unchanged.** The plane carries objects whether or not they are mined; what your application chooses to settle on chain pays the miner's fee as it always did, and nothing on the plane requires an object to be mined.

The object plane is in early access. The User Guide covers onboarding, tunnels, keys, and billing end to end; 1bsv@lightweb.net reaches a human.

8. Assumptions and limits

Quantity	Value	Basis
Worked-example BEEF object	677 B	verified: byte length of the BRC-62 example vector
Block header	80 B	structural: the Bitcoin header format, carried per BRC-135
Topic identifier	32 B, SHA-256 of the UTF-8 topic name	verified: BRC-148 topic identifier definition
Topics per submission	1–15; exactly 1 on open ingress	verified: BRC-149, submission record and admission
BEEF encodings carried	3, self-identifying by leading marker	verified: BRC-148 payload table (BRC-62/95/96)
Delivery-record framing	36 B per object (TopicID + length)	verified: BRC-149 delivery record
Per-host propagation work	one submission per interested host	structural: BRC-88 point-to-point propagation
Publisher work on the plane	one submission, independent of audience	structural: emit-once multicast
Scale target	engineered toward 1B+ TPS	stated: aspirational design target, not a measurement

Limits, stated plainly:

1. **The plane replaces propagation, not overlay judgment.** Topic managers still decide admittance at every host, and lookup services still answer queries. If your overlay's rules are wrong, the plane will deliver the wrong conclusions everywhere, faster.
2. **History bootstrap is deliberately out of scope.** Multicast carries the present. A host joining a topic, or returning from downtime, backfills through the existing overlay synchronization and lookup protocols, then rides the stream.
3. **One topic is not infinitely hot.** A single topic's sustained rate is bounded; a mechanism for spreading one very hot topic wider is reserved by BRC-148 but not yet specified. Applications expecting extreme single-topic rates should partition at the topic-naming level.
4. **"The same moment" is an architecture property, not physics.** One emission propagates down a shared tree, so subscribers hear together rather than in the order a publisher reached them; paths still differ in length. Per-flow numbering makes each subscriber's stream complete, with ordering recoverable per flow; no cross-subscriber ordering is promised, by us or by anyone honest.
5. **Early access is early.** The wire contracts (BRC-148/149) are published and stable; operating parameters such as the authenticated door's delivered-topic count are the operator's to set.

References

Overlay standards, authoritative at <https://github.com/bsv-blockchain/BRCs> (overlays/, transactions/):

BRC-22 Overlay Network Data Synchronization

BRC-24 Overlay Network Lookup Services

BRC-64 Overlay Network Transaction History Tracking

BRC-87 Standardized Naming Conventions for Topic Managers and Lookup Services

BRC-88 Overlay Services Synchronization Architecture (SHIP/SLAP)

BRC-136 Block-Anchored Overlay Synchronization (BASM)

BRC-76 Graph Aware Sync Protocol

BEEF family and wallet interface, authoritative at the same repository (transactions/, wallet/):

BRC-62 Background Evaluation Extended Format (BEEF) Transactions

BRC-74 BSV Unified Merkle Path (BUMP) Format

BRC-95 Atomic BEEF Transactions

BRC-96 BEEF V2, Txid Only Extension

BRC-100 Wallet-to-Application Interface

Multicast fabric and object-plane standards (transactions/):

BRC-124 Multicast Transaction Frame Format

BRC-126 BSV Multicast NACK Retransmission Protocol

BRC-129 IPv6 Multicast Group Address Assignments

BRC-130 Multicast Transaction Frame Fragmentation

BRC-135 Multicast Block Header Frame Format

BRC-148 Multicast Shard Domain Partitioning and the BEEF Object Plane

BRC-149 Multicast BEEF Object Frame Format

The overlay bridge: an unmodified engine on the object plane (Lightweb Inc., Paper 008): <https://1bsv.net/papers/overlay-bridge.pdf>

1BSV User Guide (Lightweb Inc.): <https://1bsv.net/papers/1bsv-user-guide.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.