DESIGN · ARCADE · MULTICAST DELIVERY

Arcade v2 on a multicast delivery fabric: one broadcast for every miner, proofs without the firehose

Arcade v2 is a Teranode-native transaction broadcaster: it accepts transactions, broadcasts them to a fleet of datahubs, and returns merkle proofs from a companion merkle-service. This paper shows how a small landing-tier bridge lets an unmodified Arcade v2 + merkle-service stack run with an IPv6 multicast fabric as its transport: its O(N) broadcast collapses to a single submit that the fabric fans out to every miner it serves, and its proof ingest is served from the landing machine itself instead of being pulled across the wide area. It closes with the follow-up work, the upstream optimizations that would tighten the integration further, and how the broadcaster tier joins the miner tier on the 1BSV network.

Author: Jeff Harris · Lightweb Inc. · jeff@lightweb.net **Paper** 007 · **Version** 02

Audience: Arcade operators and BSV application developers.

We = the 1BSV multicast fabric and its `arcade-bridge`; **you** = an operator running the Arcade v2 broadcaster stack.

The short version

- **Arcade pays two costs that scale with the network, and the fabric removes both.** Its broadcast POSTs every batch to every Teranode datahub, and its proofs come from ingesting the network's full subtree firehose. The fabric turns broadcast into one submit and delivers the firehose to the landing machine once.
- **One submit reaches every miner on the fabric.** Arcade's stock propagation is pointed at a facade that serves the datahub submission surface, converts to Extended Format where needed, and streams a single copy up one tunnel.
- **Proof ingest rides the fabric, not the wide area.** The delivery lanes feed merkle-service's own announce-and-fetch pipeline on the landing machine, so the fetch never leaves it. The fabric copy normally arrives first and front-runs the wide-area distribution; libp2p and the datahubs remain the backup path.
- **The bridge is open, and it does not fork anything.** `arcade-bridge` is Apache-2.0 and imports the shared landing-tier packages from `teranode-bridge`; Arcade and merkle-service run as unmodified upstream containers. What you pay for is fabric delivery, not the bridge.
- **It is the application-tier sibling of the miner-tier bridge.** The same fabric, edges, and tunnels that carry blocks to miners carry the data behind every proof to broadcasters, one more kind of consumer on the same multicast network.

1. The two costs Arcade pays today

Arcade v2 is deliberately light. The heavy lifting moved out into merkle-service, so an application developer can broadcast a few hundred transactions and get merkle proofs back without provisioning node-class hardware. That redesign is sound. Two costs remain, and both scale with the size of the network rather than with your traffic.

Broadcast is O(N) unicast. Arcade's propagation service broadcasts a batch by POSTing the concatenated transactions to every configured or discovered datahub in parallel. The more of the network you want to reach, the more copies you send. This is the same shape ARC had before it, and the same shape Teranode's own announce-and-pull has on the block side: the serving party's egress grows with the number of receivers.

Proofs require ingesting a firehose. merkle-service earns its proofs by subscribing to the network and ingesting every subtree as it is announced. That is Teranode-scale work by construction: at large block sizes the ingest alone approaches node-class disk and network. merkle-service pools that work so many Arcade instances share one ingest, which is the right move; but the bytes still cross the wide area once per merkle-service deployment, pulled from the announcing datahubs.

Neither cost is a defect in Arcade. They are properties of a unicast announce-and-pull world, and both disappear when the transport is a structured multicast fabric.

2. The bridge: an application-tier landing shim

`arcade-bridge` sits on the landing machine between the fabric and an unmodified Arcade v2 + merkle-service stack. It has two independent halves and holds no state of record.

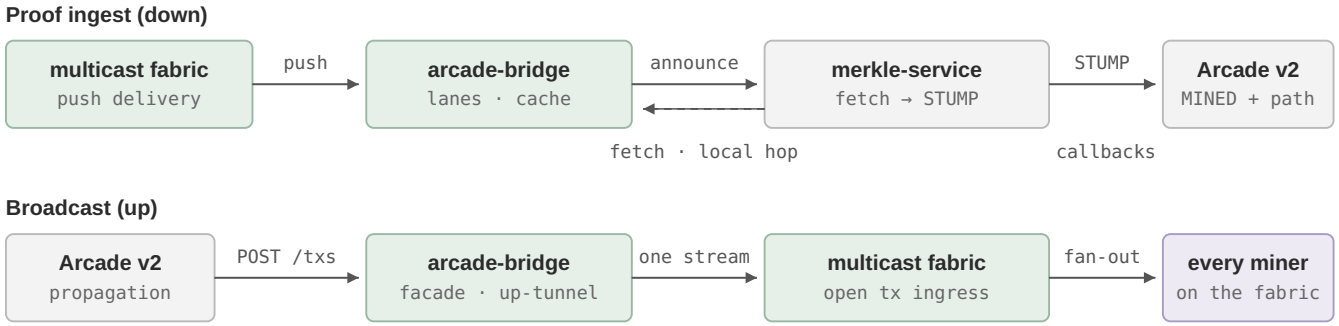

Figure 1 - the two halves of the bridge. Down: the delivery lanes feed merkle-service's own announce-and-fetch on the landing machine. Up: a facade forwards Arcade's broadcast as one stream into the fabric. Arcade and merkle-service are unmodified.

It is the sibling of `teranode-bridge`, the miner-tier landing shim that feeds an unmodified Teranode cluster. That is a deliberate split by role, not duplication. A Teranode operator is a miner: it submits subtrees and blocks and holds miner entitlements. An Arcade operator is a delivery consumer: subtrees and blocks arrive down the tunnel, transactions go up on the open class, and nothing on the broadcaster side carries miner privileges. The two bridges share their machinery (lane termination, the object cache, the retrieval plane, byte-order discipline, block-frame conversion) through a common Apache-2.0 library; the application-tier bridge imports it and never forks it.

3. Proof ingest without the firehose

merkle-service is built around announce-and-fetch: it hears a subtree or block announcement as a hash plus a URL, and fetches the bytes from that URL. The bridge already holds those bytes, because the fabric delivered them to the landing machine. So the bridge publishes the announcement merkle-service expects, on merkle-service's own announcement bus (the same surface its libp2p listener feeds), naming the bridge's local retrieval plane as the source. The fetch is served from the landing machine and never crosses the wide area a second time: for a B-byte block, merkle-service today pulls B bytes from the announcing datahubs, and on the fabric that pull is a local hop.

This is the announce-shim pattern, the same one the miner-tier bridge uses to feed Teranode. merkle-service keeps its ordinary pipeline: announce, fetch, build the STUMP, deliver the callback. Arcade's transaction lifecycle (seen, mined, compound BUMP per BRC-74) runs exactly as it would against a real datahub.

The fabric does not replace libp2p; it front-runs it. Discovery stays on, so merkle-service still hears the datahubs and can still pull from them. Because the fabric copy is local and the wide-area copies are not, the fabric announcement normally arrives first and wins merkle-service's per-object dedup; the datahub announcements arrive second and stay armed as backup. Stop the fabric feed and merkle-service falls back to libp2p and the datahubs with no lost proofs.

4. One broadcast for every miner on the fabric

The broadcast half is a facade. It serves the exact datahub submission surface Arcade's propagation already speaks, so you point Arcade at it with one configuration line and change nothing else. Each batch enters the fabric's open transaction ingress as a single stream, and the fabric fans it out to every miner-tier consumer. Your emitter cost is O(1): one submit, independent of how many miners receive it.

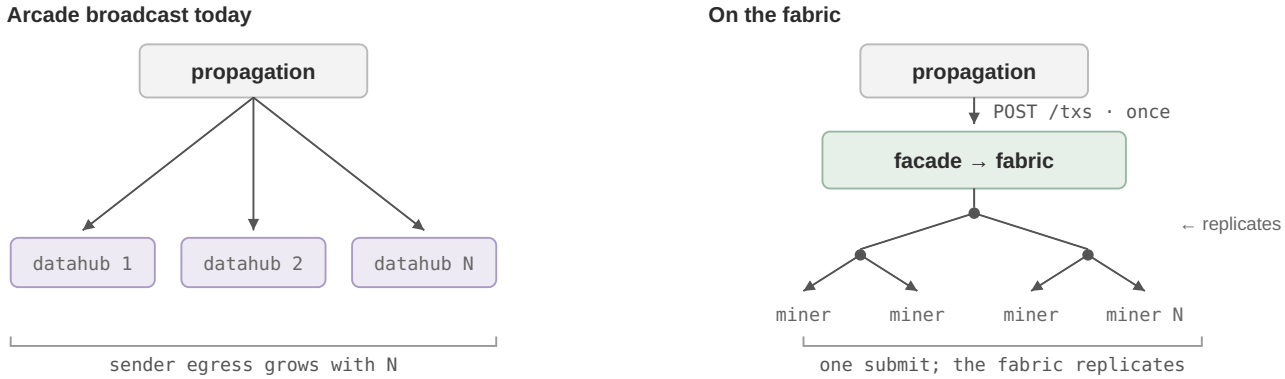

Figure 2 - broadcast fan-out. On the left the sender's egress is Σ receivers × data. On the right the sender emits once and the fabric replicates.

Miners / service providers reached	Broadcast today[1]	On the fabric[1]
10	10 copies	1 submit
100	100 copies	1 submit
1,000	1,000 copies	**1 submit**

1. Copies emitted by the broadcasting party per batch. Fabric replication cost is borne once by the network, not per sender; structural, from the fan-out shape.

One submit reaches every miner on the fabric; the sender's cost stops growing.

The fabric is Extended Format native at ingress, because that is the form a validator wants: each input carries its prior output. Arcade already converts to that form at validation time, but its propagation sends the standard serialization, so the facade re-derives Extended Format from a small cache of what it just forwarded, with an asset-service fallback (a Teranode asset API) for a parent it has not seen. A parent that cannot be found fails that one transaction with the same verdict Arcade already knows how to retry, and the rest of the batch proceeds. The facade returns per-transaction verdicts in the datahub's own grammar, so Arcade's classifier, its retry reaper, and its wallet-visible rows handle them as they would a real datahub's. The verdict itself records acceptance at the fabric ingress, one hop before any miner; it is not a datahub's verdict.

5. Deliver once per landing site

The scarce resource is the tunnel, not the landing machine: replication inside one host is effectively free. A site that runs both an Arcade stack and a Teranode cluster provisions one delivery slot and fans out locally.

Both bridges default their delivery lanes to the same canonical ports, so a site receives each object class once, whichever bridge terminates it. Two bridges cannot bind those ports on one host, by design: the collision stops a site from pulling the same bytes over two slots. When both stacks share a host, one delivery stream lands and a small tier-neutral tee (follow-up 3 in §8) fans each object out to both bridges over the loopback, where the copy costs nothing.

6. What to check before you start

#	Check	Notes
1	A provisioned delivery slot	The commercial step: a tunnel endpoint lands on your machine with subtree and block lanes
2	`arcade-bridge` beside the stack	One static Go binary, Apache-2.0, built from the public repository
3	Propagation pointed at the facade	One `datahub_urls` line in Arcade's configuration; nothing else changes
4	The announcement bus reachable	The bridge publishes on the bus the stack already runs; merkle-service needs no configuration change

Only the first row ever blocks anyone; the rest is an afternoon.

7. What it costs, and when stock Arcade is the right answer

`arcade-bridge` is Apache-2.0 open source; Arcade v2, merkle-service, and go-chaintracks run as unmodified upstream containers. The bridge has no license check, no call home, and no metering of any kind.

What is not free is fabric delivery itself: a provisioned tunnel carrying a real delivery stream is the paid relationship, the same kind of relationship a miner has. The bridge is inert without it, because a delivery shim with no lane to receive on has nothing to do. The value is not access to the bridge; it is the wide-area firehose pull and the O(N) broadcast you no longer run.

Stock Arcade over unicast is still the right answer in three cases:

1. **The network is small enough not to hurt.** With a handful of datahubs, O(N) broadcast is a few parallel POSTs and the firehose is not node-class. The costs are real but unfelt, and a tunnel cannot beat free; the case for the fabric strengthens exactly as Teranode scale arrives.
2. **N direct paths are worth their bandwidth.** POSTing to N datahubs is also N independent network paths and N verdicts from parties that will mine. Redundancy and provenance of that kind are a legitimate buy; the facade collapses it to one upstream (limit 5 in §11 states this plainly, and keeping one real datahub configured beside the facade retains an independent path).
3. **Zero external dependencies is the policy.** Declining a paid delivery relationship keeps the stack self-contained. That is a procurement position as much as a technical one, and a legitimate one rather than an obstacle to argue with.

8. Follow-up work

Both directions of the integration stand on their own. The follow-ups are enhancements, and none blocks use.

1. **A fabric-native chain-tip feed.** Arcade's chain-tip tracking still runs over go-chaintracks' own peer-to-peer path. The fabric already carries a block-header lane (BRC-135), so a feeder that pushes those headers into go-chaintracks would make the tip path fabric-native too, moving the last primary path onto the fabric.
2. **Scaling the bridge's object cache.** The stock bridge holds delivered objects in memory just long enough for merkle-service to fetch them. That window is small, but at node-class block sizes even a small window outgrows memory. The object cache is an explicit seam in the shared landing-tier library, and the miner-tier bridge fills it in its licensed delivery build with a cache backend built for that scale; the application-tier bridge inherits the same seam, so the same swap applies without touching the announce or facade surfaces.

3. **A co-located deliver-once tee.** The local fan-out of §5, for a site that runs `teranode-bridge` and `arcade-bridge` on one host. It arrives with the first such site.

9. Future upstream optimizations

These would tighten the integration and are worth raising with the upstream projects. Each has a workaround in the bridge, so none is a blocker; they turn a workaround into a first-class path.

1. **Arcade: emit Extended Format when the data is in hand.** Arcade's validator already converts each transaction to Extended Format, because it has the input source data at that point. If propagation sent that form instead of the standard serialization, the facade's re-derivation step would disappear in the common case, and miners reached directly over unicast would be spared the parent lookups. The data already exists; this is a serialization choice.
2. **merkle-service: a native fabric-ingest mode.** The bridge's proof half exists to translate the fabric's delivery into merkle-service's announcements. If merkle-service could consume the fabric directly, that translation hop would go away and the fabric would be a first-class source alongside libp2p.
3. **go-chaintracks: a header-ingest surface.** The header store accepts new headers only through its own peer-to-peer path. A thin ingest endpoint, or an explicit external-feed mode, would let the header feed in follow-up 1 (§8) be a simple shim rather than an embedded library.

10. How it fits the 1BSV solution

1BSV is one structured multicast network with several kinds of consumer on it. Miners take blocks and subtrees and submit their own; broadcasters take the subtrees and blocks their proofs are built from, and submit transactions; header and overlay consumers take exactly the classes they elect. The fabric, the edges, the tunnels, and the addressing are shared; what differs is which classes a consumer receives and whether it is entitled to the privileged ones.

`arcade-bridge` places the broadcaster tier on that network without a new substrate. Its only new surfaces are the two shims that make an unmodified application stack speak the fabric's object formats (BRC-30 transactions, BRC-143 subtrees, BRC-144 blocks): a facade for broadcast and an announce shim for proofs. The result is that an Arcade operator and a Teranode miner are two consumers of one network. A transaction a broadcaster submits enters the same fabric a miner reads, and a block a miner produces yields the proofs a broadcaster returns. The tiers compound: each miner the fabric adds extends every broadcaster's reach at no added submit cost, and each broadcaster adds demand for exactly the objects the miner tier already emits.

11. Assumptions and limits

Quantity	Value	Basis
Arcade broadcast fan-out	O(N) copies per batch	**structural**: propagation POSTs each batch to every datahub
merkle-service ingest without the fabric	the full subtree stream	**structural**: it subscribes to and ingests every announced subtree; node-class at scale
Fabric emitter cost	O(1) per submit	**structural**: one submit, the fabric replicates
Wide-area ingest displaced	one full object stream per landing site	**structural**: the fetch is served locally instead of from the announcing datahub
Bridge license	Apache-2.0, no fork	**structural**: imports `teranode-bridge` packages via module dependency

Limits, stated plainly:

1. **This paper claims the mechanism, not a peak rate.** The costs and their removal are structural; the true throughput ceilings are Teranode propagation and the fabric's transaction ingest, which are a separate body of work.
2. **Front-running is a race, and libp2p sometimes wins.** When it does, the cost is a redundant fetch and no proof is lost, because the backup path is always armed.
3. **The chain-tip path is outside the bridge.** Arcade tracks the tip over go-chaintracks' own peer-to-peer path; the header feed in follow-up 1 (§8) is the extension that brings it onto the fabric. This does not affect broadcast or proof delivery.
4. **Extended Format on the way up depends on source data.** A transaction whose parent cannot be found fails with a retryable verdict; it is never forced through.
5. **Broadcast through the facade has one upstream.** Stock propagation to N datahubs is N independent paths; with propagation pointed only at the facade, a tunnel outage parks batches in Arcade's existing retry reaper until the tunnel returns. Propagation broadcasts to every configured datahub, so keeping one real datahub beside the facade restores an independent fallback at the cost of duplicate submits. That is a configuration choice and needs no fork.

References

arcade, the Arcade v2 broadcaster (BSV Association): https://github.com/bsv-blockchain/arcade

merkle-service (BSV Association): https://github.com/bsv-blockchain/merkle-service

go-chaintracks (BSV Association): https://github.com/bsv-blockchain/go-chaintracks

`arcade-bridge` (Lightweb Inc.): https://github.com/lightwebinc/arcade-bridge

`teranode-bridge` (Lightweb Inc.): https://github.com/lightwebinc/teranode-bridge

Pushed delivery into a Teranode cluster over unicast TCP (Lightweb Inc., Paper 001): https://1bsv.net/papers/pushed-delivery-teranode.pdf

Landing in Kubernetes: pushed delivery into a Teranode cluster with no routers, no BGP, and no extra boxes (Lightweb Inc., Paper 004): https://1bsv.net/papers/kubernetes-landing.pdf

BRC-30 Transaction Extended Format (EF)

BRC-74 BSV Unified Merkle Path (BUMP) Format

BRC-135 Multicast Block Header Frame Format

BRC-143 Subtree Data Frame Format (the subtree push frame)

BRC-144 Block Frame Format (the block push frame)

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.