ANALYSIS · ASSET SERVICE · CDN OFFLOAD · TERANODE

The Asset service at scale: why fronting `DataHubURL` with a CDN extends the gossip problem instead of solving it

Teranode announces over libp2p and **pulls** the bytes from the announcing node's Asset service over unicast HTTP. That pull is an `O(N)` fan-out concentrated on one uplink, and Teranode's own deployment guidance already fronts it with a CDN. This paper walks the arithmetic up the throughput ladder from 10 k to 1 M to 1B TPS, shows where the pull model breaks, evaluates the CDN offload on its merits, and shows why it relocates the bill and the trust boundary without removing the amplification. It closes with the multicast fabric alternative, and the shared **anycast ingress with white-label co-branding** that removes the second, rarely itemised cost: every miner independently operating global transaction intake.

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

Audience: Teranode operators and engineers. **We** = the 1BSV fabric; **you** = a Teranode operator.

The short version

- **Announce-then-pull concentrates the whole network's delivery on one origin.** Every announcement triggers `N` simultaneous HTTP fetches against the announcing cluster. More Asset service pods spread the request handling, not the bytes; every copy still leaves the same cluster egress. At 1 M TPS that is ~5 Gbit/s for subtree hashes alone; at 1B TPS no origin can serve it.
- **Multiple-peer pull is the gossip blow-up wearing a different protocol.** At 1B TPS, pull by 19 peers reaches 54 Tbit/s from one origin, the same order as the 26 Tbit/s source uplink that condemns gossip.
- **A CDN moves the invoice, not the bytes.** Miner cache hit rates are near zero (unique objects, few and dispersed receivers), so the network still pays for `N` deliveries: roughly $88 k to $1.49 M per month at 1 M TPS, scaling with `N × TPS`.
- **Every CDN miss adds logical hops to the block race.** More hops means more latency, more latency means more orphans, and every orphan is forfeited revenue.
- **HTTP cannot prove absence.** A 404 is not a sequence gap; a miner discovers a missing subtree only when validation fails. The multicast fabric makes loss observable and repairable.
- **The multicast fabric emits once, at the generation rate, for any number of subscribers.** No request path to stampede, no cache to miss, no per-GB meter on the fan-out.
- **Co-branded anycast ingress deletes the duplicated intake footprint.** Miners keep their brand, their address space, and their attribution while operating none of the global entry infrastructure.

1. The shape of the problem

Teranode separates the announcement from the data. A subtree is gossipped over libp2p as `SubtreeMessage{ Hash, DataHubURL }`; a block likewise carries a `DataHubURL`. Both are small. The **bytes** (subtree hashes, missing transactions, the block itself) are fetched over unicast HTTP from the announcer's Asset service.

That is two scaling surfaces stacked on each other:

- **A gossip control plane.** libp2p gossipsub: mesh amplification, non-deterministic paths, no ordering, no gap signal. Cheap per message, and not where the bytes are.
- **A unicast pull data plane.** Every announcement triggers an `N`-way HTTP stampede against **one** origin: egress `N × object`, compressed into the propagation window, at the moment orphan risk is being set.

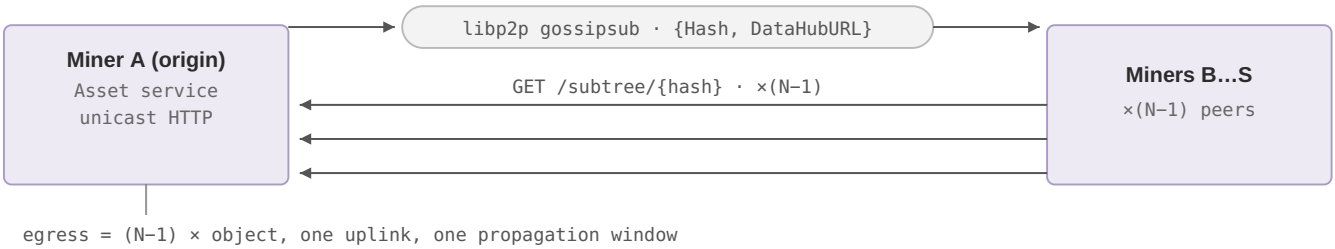

Figure 1 - announce over gossip, pull over unicast. The announcement fans out across the mesh; the bytes all come off the announcing cluster's egress.

This reads as a fine architecture today because the numbers are trivial today; at 10 k TPS the whole thing costs an origin about 50 Mbit/s. The ladder is where it stops being fine.

2. Grounding the arithmetic

Let T be transactions per second, S the mean transaction size, and $D = T \times S$ the generation rate: the bytes the network must move, continuously, before any distribution overhead.

Transaction size. A planning mix of 90 % small transfers at 250 B and 10 % larger transactions at 1 000 B gives $S = 325$ B. At T = 1 billion, D = 325 GB/s, which is **2.6 Tbit/s**. Alongside the transaction itself, each transaction contributes one 32 B node hash to a subtree, so the full object stream a peer needs is **357 B/tx**.

The distribution floor. Delivering D to R receivers requires at least $R-1$ copies to cross the network: total link work $\geq (R-1) \times D$. No architecture beats that floor; they differ only in **where it lands**:

- **Multicast.** The source emits once; uplink stays at D regardless of R. Replication happens at branch points, and no link carries more than one copy.
- **Gossip.** Each node forwards to a fanout g, so the source uplink is $g \times D$. At $g = 10$ and 1B TPS, **26 Tbit/s** leaves the source, and mesh overlap adds roughly 30–40 % of redundant transmissions above the floor.
- **Unicast pull.** The entire $(R-1) \times D$ lands on the origin's single uplink, compressed into the propagation window.

In hardware terms, D alone at 1B TPS is 26 saturated 100 GbE ports, or four 800 GbE ports, before any architecture multiplies it.

Receiver set. We take $R = 20$ block producers, so $N = 19$ pulling peers. The Asset service is a public HTTP endpoint and other parties pull the same paths, so 19 is a deliberate **floor** on the fan-out, not an estimate of it.

The framing everything below depends on: pull does not create extra bytes relative to the floor. It **relocates** them, all of $(R-1) \times D$ onto one origin's uplink, plus second-order duplication wherever the N unicast paths share backbone segments. Multicast does not cheat the floor; it declines to put all of it in one place.

3. Walking the ladder

3.1 Bandwidth

Two bounds per rung. **Lower**: peers already hold the transactions and pull only subtree hashes (32 B/tx). **Upper**: peers also pull the transactions they are missing, approaching the full 357 B/tx object stream.

Aggregate TPS	Subtree stream (32 B/tx)	Origin egress, N = 19 (lower)	Origin egress, N = 19 (upper)
10 k	2.56 Mbit/s	48.6 Mbit/s	543 Mbit/s
100 k	25.6 Mbit/s	486 Mbit/s	5.43 Gbit/s
1 M	256 Mbit/s	4.86 Gbit/s	54.3 Gbit/s
10 M	2.56 Gbit/s	**48.6 Gbit/s**	543 Gbit/s
100 M	25.6 Gbit/s	486 Gbit/s	5.43 Tbit/s
1B	256 Gbit/s	**4.86 Tbit/s**	**54.3 Tbit/s**

At 1 M TPS a single origin spends ~5 Gbit/s, half a 10 GbE NIC, purely to hand nineteen peers the same subtree hashes. At 10 M TPS it is past a 25 GbE NIC. At 1B TPS it is 4.86 Tbit/s from one origin (six saturated 800 GbE ports) for a stream generated at 256 Gbit/s; with missing transactions included, 54.3 Tbit/s.

Horizontal scale inside the origin does not change this. Multiple Asset service pods divide the request handling and multiply the §3.2 concurrency ceilings, but they share the cluster's uplink and the same object store: the egress total is set by the topology, not by the pod count. The same holds for replicas of any other service placed in front of the bytes; $N \times \text{stream}$ still leaves one cluster.

That is the sentence this table exists to produce. Multiple-peer HTTP pull at 1B TPS lands at the same order as the 26 Tbit/s gossip blow-up computed in §2, and above it. Unicast pull *is* gossip amplification with extra steps: the redundancy moved from the mesh onto the origin's uplink, concentrated rather than distributed, and exactly one operator pays for it.

3.2 Request rate and concurrency: the ceiling that binds first

Bandwidth is not what fails first. The Asset service bounds its heavy subtree paths with per-method concurrency semaphores, deliberately small by default:

Setting	Default	Governs
`asset_concurrency_get_subtree_data`	**2**	parallel subtree-data fetches
`asset_concurrency_get_subtree_data_reader`	**4**	parallel subtree-data streaming reads
`asset_concurrency_subtree_data_create`	**4**	on-demand subtree-data file creation
`asset_subtreeDataStreamingConcurrency`	**2**	parallel chunk workers within one stream

These are repository-level semaphores, not rate-limit middleware (**no peer tier is exempt**), and the create path rejects excess requests with **HTTP 503** rather than queuing. An N-way stampede is precisely the shape that hits the cap.

Take a 2^{20}-leaf subtree (~1.05 M transactions: ~34 MB of hashes, ~341 MB with transaction data; Teranode's documentation notes these endpoints "can emit hundreds of MB per request uncompressed"):

Aggregate TPS	Subtrees/s	Pull req/s (N = 19)	Budget per request at concurrency 4	Implied per-stream rate (34 MB)
1 M	~0.95	18	221 ms	1.22 Gbit/s
10 M	~9.5	181	22 ms	12.2 Gbit/s
100 M	~95	1 812	2.2 ms	122 Gbit/s
1B	~954	18 120	0.22 ms	**1.22 Tbit/s**

The right-hand column is what a **single stream** must sustain for the default concurrency to keep up. It is not reachable, and raising the semaphore converts a concurrency failure into the §3.1 bandwidth failure plus the memory pressure Teranode's docs warn about (the create path "can OOM under load"). Subtree size is not a lever: smaller subtrees mean proportionally more requests, and the aggregate rate is unchanged.

A per-IP heavy-endpoint limiter (`asset_httpHeavyRateLimit`, default 10 req/s) exempts verified mining peers, but only peers listed in `asset_peerAuthAllowlist`, which is **empty by default**, with `BlocksReceived > 0` and sufficient reputation. The pull model's viability thus rests on every operator maintaining an allowlist of every other operator: an `O(N²)` trust-configuration burden that grows with exactly the thing the network wants to grow.

3.3 The burst factor

None of this is a steady stream. The pull is `N` synchronised GETs triggered by one gossip message inside one propagation window. Demand is perfectly correlated across receivers, so sizing against the mean under-provisions at exactly the latency-critical event: the block race itself.

4. The CDN reflex, stated at its strongest

The Asset service serves **immutable, content-addressed** objects over plain HTTP: `/subtree/{hash}`, `/subtree_data/{hash}`, `/block/{hash}`. That is the textbook CDN workload. Put a CDN in front, publish the CDN hostname in `DataHubURL`, and origin egress collapses from `N × stream` to one fill per PoP miss; with an origin shield, to roughly one fill total.

This is not a strawman. **Teranode's own deployment guidance mandates it.** The Asset service documentation states its in-process mitigations "are **not** sufficient on their own for public-facing deployments" and operators "MUST front the asset service with a reverse proxy (nginx, HAProxy, Envoy, CloudFront, Cloudflare, etc.)". The CDN is the documented posture, not a hypothetical.

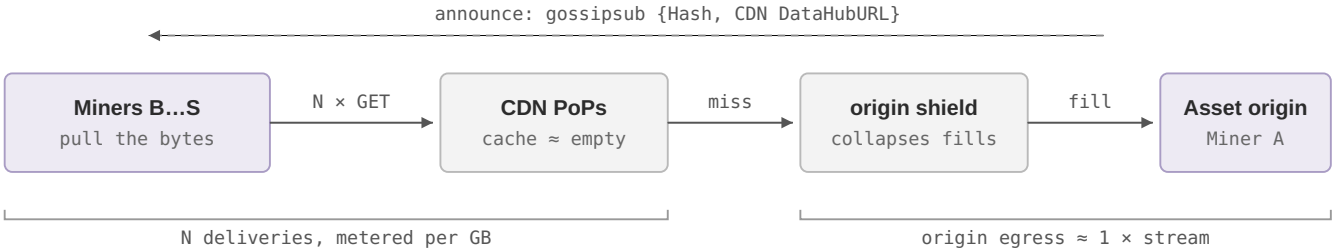

Figure 2 - the CDN-fronted pull. Origin egress does drop. Everything else about the topology is unchanged.

And it is genuinely attractive: no new protocol, no new peer relationships, no Teranode change, an operational model every infrastructure team understands, and a vendor with an SLA. If the argument against it were weak, it would be the right answer.

5. Why it does not hold

5.1 The cache has nothing to cache

A CDN converts one origin fetch into many edge deliveries when an object is **popular**: requested many times, over a long life, through few PoPs. Subtree objects violate every condition:

- **Unique.** Every object is new; the working set is entirely cold.
- **Single-use in time.** Fetched once per peer inside a few-second window, then never again. The cache fills and immediately drains.
- **Few, dispersed receivers.** The per-PoP hit-ratio ceiling is `(k−1)/k`, where `k` is receivers homed to that PoP. With ~20 producers deliberately spread across ~20 metros (the *desired* topology), `k ≈ 1` and the hit ratio approaches **zero**.

A CDN's fan-out multiplier is a function of receivers-per-PoP. **Miners are the anti-pattern for it.** Tiered caching and an origin shield do collapse origin fills (a real win for origin protection), but you still pay egress on all `N` deliveries plus the mid-tier fill. You have bought a proxy, not an amplifier.

5.2 The bytes do not disappear, the invoice moves

The `N × stream` never stopped existing; it changed billing entity, from transit commits to a per-GB retail meter. Network-wide, at `N = 19` and 30 days/month:

Aggregate TPS	Egress/mo (subtree lower bound)	Cost band[1]	Egress/mo (full object stream)	Cost band[1]
10 k	15.8 TB	$79 – $1.3 k	176 TB	$0.9 k – $15 k
100 k	158 TB	$0.8 k – $13 k	1.76 PB	$8.8 k – $149 k
1 M	1.58 PB	$8 k – $134 k	17.6 PB	$88 k – $1.49 M
10 M	15.8 PB	$79 k – $1.34 M	176 PB	$0.9 M – $14.9 M
100 M	158 PB	$0.8 M – $13.4 M	1.76 EB	$8.8 M – $149 M
1B	1.58 EB	$8 M – $134 M	17.6 EB	**$88 M – $1.49 B**

*1. Network-wide per month: the sum of origin egress across all producers, at ~$0.005/GB (large-commit) to ~$0.085/GB (list). Each miner bears the share of blocks it produces, but must still provision the **full** `N × stream` burst for the moments it is the one serving.*

At 10 k TPS the bill is a rounding error, which is why the architecture reads as sound. At 1 M TPS it is a real line item; at 10 M a budget; at 1B not a price anyone quotes.

The slope is the actual argument. The CDN column scales with `N × TPS`; the fabric column with `TPS` alone. Growing the miner set, the thing every participant claims to want, makes the transmission bill monotonically worse for **every existing miner simultaneously**; under emit-once it changes nothing. An architecture whose cost is superlinear in participation prices participation out.

The fair objection: nobody pays list at exabyte volume; you would negotiate a private rate. True, and it is the point: at that volume you are negotiating for a **private network**. You have re-derived the multicast fabric, rented, metered per gigabyte, with no gap detection and no in-network replication.

5.3 The miss path is the block race

Propagation lives on a hop budget. A distribution tree that reaches every receiver within a few hops keeps delay bounded and predictable; every hop added past that budget compounds orphan risk, stretches the latency tail, and fractures the miners' shared view of the chain.

Now count the CDN path's logical hops. Nearly every miner fetch is a miss (§5.1), and a miss costs client to PoP, PoP to mid-tier shield, shield to origin, and back again: three round trips stacked in series where the direct pull had one, and where a multicast tree has a bounded forward path with no request leg at all. Request collapsing at the PoP serialises concurrent misses behind one origin fetch, which, given §3.2, may be the fetch that receives the 503. TLS handshakes, per-tier queueing, and retry timeouts ride on top.

The revenue arithmetic is direct. Orphan rate is a function of propagation delay, and an orphaned block forfeits the entire reward and its fees. A CDN in the consensus path converts added latency into lost income at exactly the moment the infrastructure is supposed to be earning it, while optimising for a cache-hit case that structurally barely occurs. For contrast, the 1BSV fabric's measured overhead **above the physical RTT floor** is ~250 µs p50 / ~600 µs p99 on a geo lab with configured ocean delay: delivery rides the speed of light, not a cache-fill round trip.

5.4 Loss stays invisible

HTTP gives you a 404, a 503, or a timeout, not a **sequence gap**. No NACK, no repair horizon, no bounded recovery, and no way to know what you did *not* receive. A missing subtree is discovered when validation fails: after the fact, where the cost is highest.

This property, **discoverability of loss**, is one a CDN cannot provide at any price, because it is not a bandwidth property. Per-flow monotonic sequence numbering makes absence a first-class, observable event: gap → targeted NACK → bounded retransmit from a short-TTL cache → verified reintegration. Repair traffic is proportional to loss experienced, not to `N`, and silent divergence between two operators' views becomes structurally impossible.

5.5 The gossip is still there

The announce plane is untouched: same gossipsub, same amplification, same non-determinism. The CDN changes **who serves the stampede**, not that a stampede is how data moves.

This is the crux. Fronting the Asset service **extends the gossip network into a third-party HTTP tier**. It adds a plane rather than removing one: a gossip control plane, a unicast pull data plane, *and* a CDN, with the failure modes of all three composed. The amplification has not been eliminated; it has been outsourced, and given a monthly invoice.

5.6 A third party is now in the consensus path

Direct consequences of Figure 2:

- **Availability coupling.** Block propagation now depends on a vendor's config, WAF rules, and rate limits, and an outage is correlated across every miner on the same provider: fleet-wide and simultaneous, the precise failure shape consensus infrastructure must avoid.
- **TLS terminates at their edge.** Consensus data is decrypted by a third party by construction, and the provider observes which peers pull what, when, and from where: a timestamped map of the miner graph.
- **The auth model and the cache model pull against each other.** Teranode's peer authentication signs each request (method, URI, body digest, ±10 s freshness window, replay cache), and tier elevation is evaluated **at the origin**, keyed by peer ID. A cache hit never reaches the origin, so on exactly the requests the CDN absorbs, the ban-list, the reputation gate, and the tiered limiter do not run.
- **There is nobody to NACK.** Recovery is a support ticket.

5.7 What a CDN legitimately buys

To be clear about where the line falls: a CDN in front of the Asset service is **correct and worth doing** for origin protection, TLS offload, and geographic termination of explorer, dashboard, and general API traffic. Teranode's guidance is right for those reasons. It is not a distribution architecture for consensus data. The two are different jobs, and the first being real does not make the second work.

6. The second bill: global transaction intake

Everything above concerns the **download** half of the path. There is a second cost, structurally larger, that the CDN discussion cannot touch at all.

Under the current regime every Teranode miner is its own global intake point. A wallet submits where it can reach, and submit latency determines where transaction flow lands. Being close to senders wherever they are is not an optimisation a miner may defer; it is competitive, which makes it mandatory.

There are several ways to get there, and they are not equal:

Approach	Cost profile	What you give up
Build your own global entry network	Highest: sites, hardware, transit, routing and DDoS expertise per region	Nothing, if executed well, everywhere, continuously
Rent equivalent reach from providers	Recurring per-site and per-traffic fees	Some control; outage correlation with everyone on the same provider
DNS-based geographic steering	Cheap	Slower failover, coarser proximity, resolver-cache exposure
Fewer regions	Cheapest	Submit latency wherever you are absent; that flow lands elsewhere

Whichever path a miner picks, the goal is identical, and at competitive quality it runs to **tens of thousands of dollars per month and up, per miner**, with every site sized for its peak rather than its share of the mean. Cheaper rungs exist; each surrenders latency or reliability, which is to say flow. And the network-level result is the same regardless: `N` miners independently building or renting **the same coverage of the same senders**, duplicated `N` times over, each footprint at poor utilisation.

A CDN does nothing here. A CDN accelerates **downloads**; submission is **upload to an origin**, where CDNs are weakest and most expensive. An operator who "solves" transmission cost with a CDN has addressed the smaller of their two structural costs.

7. What the multicast fabric does instead

Emit once. A transaction maps deterministically to a shard group (`groupIdx = f(TxID)`), is published **once** into the multicast fabric, and the fabric delivers it to every subscriber. Source cost is `O(1)` in subscriber count. There is no request, so there is no stampede, no hit ratio, no origin shield, no concurrency semaphore, and no per-GB meter on the fan-out. Replication happens at branch points: the property a CDN approximates through caching, and misses because caching only works when receivers are many and co-located.

Push delivery deletes the pull. Subtrees arrive as **BRC-143** frames with the merkle root in-band; blocks as **BRC-144** with the coinbase in-band, plus height and the coinbase merkle path: everything block validation needs with no follow-up fetch. `DataHubURL` leaves the steady-state path entirely. The CDN question is not answered; it is **deleted**, because there is no request to offload.

Loss you can see. Per-flow monotonic sequence numbers: gap → targeted NACK → bounded retransmit from a short-TTL cache. Repair traffic is proportional to observed loss, not to `N`.

Sharding is the headroom. Increase the shard count and each group is an independent flow: the multicast fabric scales horizontally without any single link getting hotter, and a delivery edge carries only what its consumer elected.

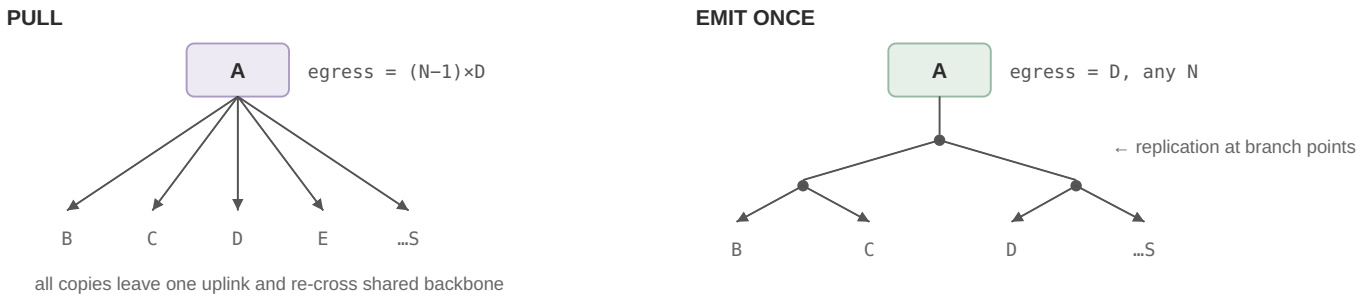

Figure 3 - the same delivery, two placements of the same `(R−1) × D` floor. Multicast does not beat the floor; it declines to put all of it on one uplink.

8. Shared anycast ingress with white-label co-branding

The commercial answer to §6, and the identity answer to the whole paper.

8.1 Two tiers

- **Tier 1 (DNS, default).** The miner's own domain points at the 1BSV anycast ingress. No routing infrastructure, no capital outlay, no sites. The submitter resolves and connects to **the miner's hostname**.
- **Tier 2 (prefix, premium).** 1BSV announces a **miner-owned address block** as anycast from its ingress edges, with cryptographically authorised routing (RPKI). Packets are addressed into the miner's own address space, announced on its behalf, from locations the miner does not have to operate: identity at the **routing layer**, not merely in DNS.

8.2 The design property that makes it cheap

A co-brand is **never a source on the multicast fabric and never rides in the frame**. The admitting ingress attributes it as a **per-domain metric at admission**, for the miner's metrics and billing, and delivery to every subscriber stays uniform.

That one constraint is what makes the offer inexpensive: identity **without** fragmenting the fabric into per-miner trees, without per-miner delivery paths, without frame-format changes, and without per-brand replication cost. The hundredth co-brand costs what the first did: a DNS record or a prefix announcement, and a label on a counter.

8.3 Identity through the entire delivery process

Follow one transaction:

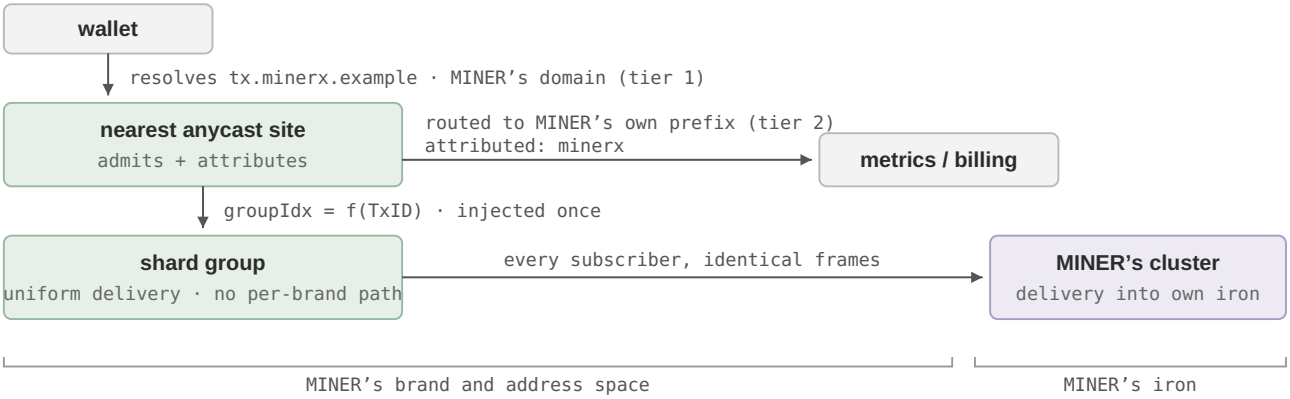

Figure 4 - brand at ingress, attribution in the metrics, uniformity on the wire, delivery into the operator's own cluster.

The submitter sees the miner's hostname and certificate. Under tier 2 the packets are addressed to the miner's prefix. Admission records the transaction against that miner. The frame on the fabric is identical to every other frame, so the multicast fabric costs nothing extra to carry it. And delivery lands in the miner's own cluster, over its own tunnel, into its own hardware.

The one-line contrast with the CDN model: a CDN puts the miner's brand on a **download URL**, the half of the path that does not determine where flow goes. Co-branded anycast ingress puts it on the **submission** path, which is the half that does.

9. Side by side

This section stands alone.

Axis	Self-served pull	CDN-fronted pull	1BSV multicast fabric
Origin egress	`N × stream`	~`1 × stream` origin, `N × stream` billed at edge	`1 × stream`
Slope in participants	linear in `N`; worse as the network grows	linear in `N`; worse as the network grows	**flat**; `N`-independent
Transmission cost @ 1 M TPS	own transit at 4.9 – 54.3 Gbit/s sustained	$88 k – $1.49 M / month network-wide	one delivery flow, sized to elected traffic
Transmission cost @ 1B TPS	4.86 – 54.3 Tbit/s from one node	$88 M – $1.49 B / month network-wide	emit-once at generation rate; shard horizontally
Logical hops in the race	one round trip per object	three stacked round trips on every miss	bounded forward path; no request leg
Concurrency ceiling	2–4 default semaphores; 503 on overflow	unchanged on every miss	no request path; no ceiling to hit
Global intake footprint	duplicated per miner, sized for peak	**unchanged**; CDNs do not accelerate upload	shared; the miner operates none
Loss visibility	none; 404/timeout, discovered at validation	none, plus a cached 503	sequence gap → NACK → bounded repair
Trust boundary	your infrastructure	vendor terminates TLS, sees the miner graph	your prefix, your cluster, uniform frames
Identity	your domain, on the download half	your domain, on the download half	your domain **and prefix**, on the submit half

What a miner stops paying for: a duplicated global intake footprint and its peak-sized capacity; a CDN contract and its per-GB meter; Asset-service fan-out capacity for `N` simultaneous streams; DDoS absorption on an open submission endpoint; certificate lifecycle at every edge.

What stays entirely theirs: their brand, their address space, their attribution and metrics, their cluster, their hardware, their block assembly, their consensus role. Nothing in this model asks a miner to become a tenant of anyone's chain.

What it costs on the Teranode side: accepting pushed subtrees and blocks instead of pulling them is a real change (new ingest paths for the BRC-143/144 push frames, idempotent dedup, reconnect tolerance), specified separately.

10. Assumptions and limits

Quantity	Value	Basis
Transaction size	325 B weighted mean	**stated**: 90 % × 250 B + 10 % × 1 000 B
Subtree entry	32 B/tx	**structural**: one node hash per transaction
Full object stream	357 B/tx	**derived**: transaction plus subtree entry
Pulling peers	`N = 19` (`R = 20` producers)	**stated**: deliberate floor; public pullers excluded
Subtree size	2^{20} leaves ≈ 1.05 M tx	**stated**: scales §3.2 request rate, not aggregate load
CDN egress price band	$0.005 – $0.085 / GB	**modelled**: large-commit to list; 30-day months
Global-intake cost	tens of $k/month and up	**modelled**: order of magnitude at competitive quality
Asset concurrency defaults	2 / 4 / 4 / 2	**verified**: Teranode `settings/asset_settings.go`
Heavy-endpoint limit	10 req/s, mining peers exempt	**verified**: `asset_httpHeavyRateLimit`
Fabric overhead above RTT	~250 µs p50 / ~600 µs p99	**measured**: 1BSV geo lab, configured ocean delay

Three limits, stated plainly:

1. **The price bands are modelled, and the conclusion does not depend on them.** Even at the best committed rate the CDN column still scales with `N × TPS`. The **slope** fails, not the constant; negotiating the constant down buys a rung on the ladder, not the ladder.
2. **Capping the fan-out at 19 is conservative.** The Asset service is a public endpoint; exchanges, overlay services, and explorers pull the same paths. Every table above understates the pull load by exactly the number of non-producer pullers.
3. **The `(R−1) × D` floor is real and multicast does not beat it.** The claim here is narrower: emit-once holds the **source uplink** flat, replicates at branch points instead of re-crossing shared backbone segments, and removes the per-GB meter and the third party from the consensus path. The bytes do not vanish.

References

Craig Wright, *Multicast as the Only Viable Architecture for Billion-Transaction Networks* — https://singulargrit.substack.com/p/multicast-as-the-only-viable-architecture

Craig Wright, *Multicast Within Multicast: Anycast, Sharded Resends, and Hierarchical Distribution for Transaction and Block Propagation* — https://singulargrit.substack.com/p/multicast-within-multicast-anycast

Multicast frame standards, authoritative at https://github.com/bsv-blockchain/BRCs (transactions/):

BRC-124 Multicast Transaction Frame Format

BRC-126 Multicast Transaction NACK Retransmission Protocol

BRC-128 Multicast Extended Transaction Frame Format

BRC-129 IPv6 Multicast Group Address Assignments

BRC-130 Multicast Transaction Frame Fragmentation

BRC-131 Multicast Block Announcement Frame Format

BRC-132 Multicast Subtree Data Frame Format

BRC-133 Multicast Coinbase Transaction Frame Format

BRC-134 Multicast Anchor Transaction Frame Format

BRC-142 Multicast Transaction Bundle Frame Format

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

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

Teranode — https://github.com/bsv-blockchain/teranode (Asset service documentation and `settings/asset_settings.go`)

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

---

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