DESIGN · MULTICAST · REPAIR

Loss on the Internet: the network that notices, and asks for exactly what it lacks

Multicast has no acknowledgments, so a frame lost at a branch of a tree is lost to every receiver below it and nothing upstream knows. The fabric is built on the assumption that this happens all the time, and it notices. This paper walks through how, one picture at a time: the number every frame carries from the door, the gap a receiver sees the instant the next frame arrives, the small request it sends to a cache by name, the three answers a cache can give, how a receiver finds caches and walks from one to the next, how loss that happened before any nearby cache is fetched from one hop upstream, why a resend can never pose as a new frame, why a fragment is repaired like any other frame, and what a tunnel changes, which is nothing. It closes with the one blind spot, stated plainly, and with what a consumer sees.

Author: Jeff Harris · Lightweb Inc. · jeff@lightweb.net **Paper** 015 · **Version** 01

Audience: network engineers and implementers who assume multicast means best effort, and anyone deciding whether a lossy path can carry this.

We = Lightweb Inc., author of this architecture and operator of the first fabric built on it; nothing here depends on that operator. **You** = a receiver, or an engineer sizing a path the fabric crosses.

Every figure uses the same colors: green for the fabric's trees, frames and caches, purple for receivers and consumers, gray for the Internet. A teal line is a request or a resend; a red cross is a loss.

The short version

- **Every frame is numbered at the door.** The ingress stamps a flow key and a sequence number before anything is replicated, so every copy anywhere is the same numbered frame.
- **A gap is seen, not guessed.** Sequence 3 followed by 6 means 4 and 5 are missing, known the instant 6 arrives, with no timer.
- **The receiver asks by name.** A request of a few dozen bytes names the flow and the range, and goes to a cache the receiver already knows.
- **A cache answers at once: here it is, I do not have it, or wait.** A miss costs no time; the receiver asks the next cache immediately.
- **Loss upstream of every nearby cache is fetched once from one hop up.** A cache that misses may ask one cache nearer the source, re-cache the answer and resend it, and the relay is bounded to that single hop.
- **A resend cannot masquerade.** It carries the same flow key and sequence number as the frame it replaces, so a receiver that already has it drops it, and a forged resend matches no flow.
- **A tunnel changes nothing.** The repair protocol does not know what link lost the frame; the loss rate of the path sets only how often it runs. A consumer on a unicast lane never sees it run.

1. Best effort, and what it means on a tree

A unicast connection acknowledges what it received and resends what it did not. A multicast tree cannot: a source sending to a thousand receivers cannot hold a thousand conversations, and if it did there would be no point in multicast. So a datagram dropped on a link is dropped for every receiver below that link, silently. That is not a defect of a tunnel or of the Internet. It is the nature of a tree, and any design that carries value over one has to treat loss as ordinary and recover from it without the source's help.

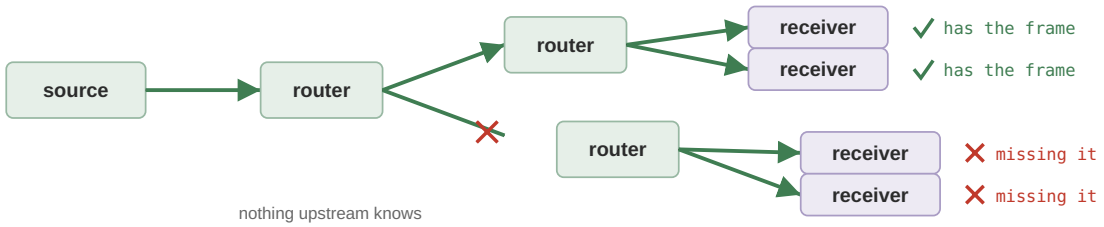

Figure 1 - a frame dropped below one branch. Every receiver under that link is missing the same frame, and nothing upstream knows.

2. Numbering at the door

Repair begins before any frame is lost, at the ingress. When a transaction or object is admitted, the ingress stamps its frame with two things: a flow key, which identifies the stream of frames from that ingress to that group, and a sequence number, which counts the frames in the flow from one upward (BRC-124). The stamp is applied once, before the frame is replicated anywhere, so every copy of that frame on every branch of every tree carries the same key and the same number. A cache stores the frame under them, a receiver tracks the flow by them, and a request names them. Nothing downstream ever renumbers.

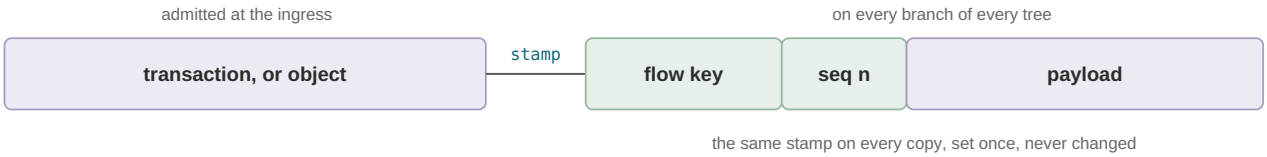

Figure 2 - the stamp. A flow key and a sequence number are set once at the door and travel unchanged on every copy.

3. A gap is seen, not guessed

A receiver keeps, for each flow it is receiving, the highest sequence number it has seen. When the next frame arrives with a number more than one above it, the numbers in between are missing, and the receiver knows exactly which. There is no timer to expire and no estimate of what should have arrived; the next frame names the gap. The only wait is a short, deliberately randomized hold before asking, so that a thousand receivers under the same lost branch do not all ask the same cache in the same microsecond.



Figure 3 - the sequence line. Frames 4 and 5 are missing, and the receiver knows it, and knows their numbers, when 6 arrives.

Not every jump is a loss. An ingress that restarts begins a new count, and a jump far larger than any plausible loss, or one that the flow's own rate says could not have been carried in the time elapsed, is read as a new emitter rather than as thousands of missing frames. The receiver re-baselines the flow and asks only for the few frames a real outage of that length could have lost.

4. Asking by name

The request is small: a few dozen bytes naming the flow key and the first and last missing numbers, sent over plain unicast to a cache the receiver knows about (BRC-126). It carries no payload and asks for nothing the receiver cannot already name. The cache can answer in one of three ways. It can resend the frames. It can say at once that it does not have them, in which case the receiver loses no time before asking elsewhere. Or it can say wait: the request hit a rate limit for that gap, that flow or that group, which usually means a repair for exactly that gap is already on its way, so the receiver holds briefly and asks the same cache again rather than moving on.

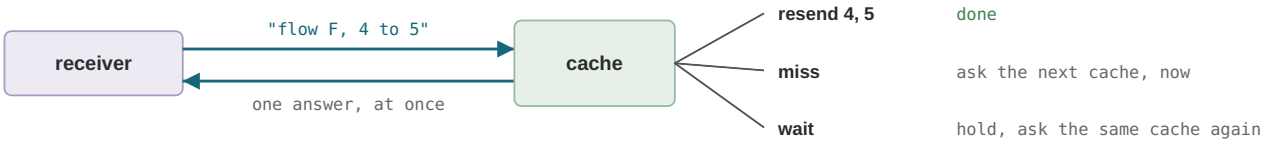

Figure 4 - the request and its three answers. A miss is immediate, so a receiver never waits on a cache that cannot help.

How the resend travels is the operator's choice. A cache may send the frames back to the requester alone over unicast, which costs one copy and reaches exactly who asked; or it may resend them into the group, where every receiver under the same lost branch gets them at once and the others, which already have them, drop the duplicates by number (§7). The protocol allows both, and a receiver treats either as a repair.

5. Caches, and finding them

A cache sits beside the trees, receives every group it serves, and keeps each frame for a bounded window keyed by its flow and number. Receivers learn which caches exist from an announcement the caches themselves send on a control group (BRC-126), carrying each cache's address, its distance from the source in network terms, and its preference among caches at the same distance. A receiver asks the nearest, most preferred cache first, and on a miss moves to the next in order, with no delay between. The walk is ordered so that the same frame is served from as close to the receiver as it exists.

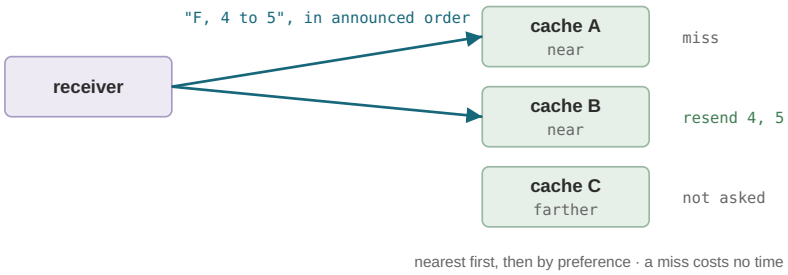

Figure 5 - walking the caches. The receiver asks in announced order and stops at the first cache that has the frames.

6. Loss upstream of every cache you know

A receiver's walk is bounded by the caches it has heard announced, which are the caches of its own region. If a frame was lost on a link between regions, it never reached that region at all, and every cache the receiver can name is missing exactly the same frames. The walk would end in misses. For that case a cache may be configured to relay: on a miss, it answers miss at once, then asks one cache nearer the source for the frames, re-caches them when they arrive, and resends them into its own region, where the receiver's gap closes on its own. The relayed request is marked, and a marked request is never relayed again, so the chain is one hop long by construction and a region's caches cannot be used to pull frames across the whole fabric.



Figure 6 - the one-hop relay. The regional cache misses, asks one cache upstream across the Internet, re-caches the answer and resends it; the marked request is never relayed again.

7. Why a resend cannot masquerade

A resent frame is the frame: the same flow key, the same sequence number, the same bytes. A receiver that already has that number drops the copy. One that was missing it fills the gap and continues. Because the key and number are set at the ingress and never changed, a cache can serve a frame it did not originate, from any region, and the receiver cannot tell the difference, which is the point. A forged resend, by contrast, has to name a flow and a number the receiver is expecting, and then carry content that verifies: a transaction that checks by its signature, an object that checks by its proofs. A frame that does neither is dropped at the receiver whether it arrived on the tree or as a repair.

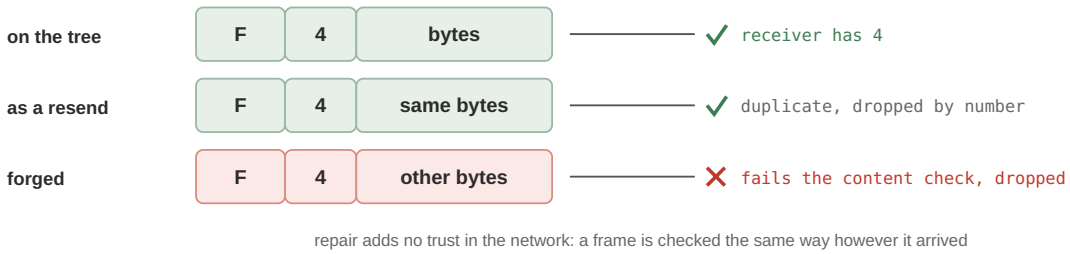

Figure 7 - a resend is the same numbered frame, so a duplicate is dropped by number and a forgery by content. Repair adds no trust in the network.

8. Fragments are repaired the same way

A large object crosses the fabric as fragments, and each fragment is a frame with its own sequence number in the same flow, as *Frames on a 1500-byte Internet* sets out. A lost fragment is therefore a gap like any other: seen at the next number, asked for by name, served from a cache, and dropped if it arrives twice. The receiver's reassembly buffer waits for the fragment to be repaired before it declares the object complete, and an object whose fragments cannot be recovered within the window is abandoned whole rather than delivered short. Nothing in the repair protocol knows that a fragment is part of anything.



Figure 8 - a lost fragment is one missing number. Repair closes it, and the reassembly buffer completes the object.

9. What the tunnel changes: nothing

The repair protocol sees flows and numbers. It does not see links, and it cannot tell whether frame 44 was lost on a tunnel across the Internet, on a cross-connect, or on a cable in a rack. What a lossy path changes is only how often repair runs, and that is arithmetic. Take a receiver taking 10,000 frames a second from a flow that crosses a link losing one frame in a thousand. It sees about ten gaps a second, sends about ten requests a second of a few dozen bytes each, and receives about ten resent frames a second, which at full size is on the order of fifteen kilobytes a second, or about a tenth of a percent of the flow. The work of repair is proportional to the loss, and the loss on a well-chosen Internet path is small.

Loss on the path	Gaps a second at 10,000 frames/s	Request bytes a second	Resent bytes a second, 1500-byte frames	Share of the flow
1 in 10,000	1	about 60	about 1,500	0.01 %
1 in 1,000	**10**	about 600	**about 15,000**	**0.1 %**
1 in 100	100	about 6,000	about 150,000	1 %

1. Frame rate, frame size and loss rates stated; request size taken as a few dozen bytes; one request and one resend per lost frame.

Repair costs what the loss costs, and nothing more; a path that loses little is repaired little.

10. The blind spot, stated

A gap is seen at the next frame, so the last frame of a flow that then falls silent has no successor to reveal it. If frame 50 is lost and the ingress sends nothing more on that flow, no receiver knows 50 existed until the flow resumes, and if it resumes after the cache's window has passed, 50 is gone. On a fabric whose flows are continuous this is a narrow exposure: the next frame is milliseconds away, and the window is long by comparison. It is not zero, and the design does not pretend it is. The remedy, where an operator wants it, is a periodic marker on a quiet flow that tells receivers the current number, so silence itself can be checked; it is a small addition and not part of the base protocol.

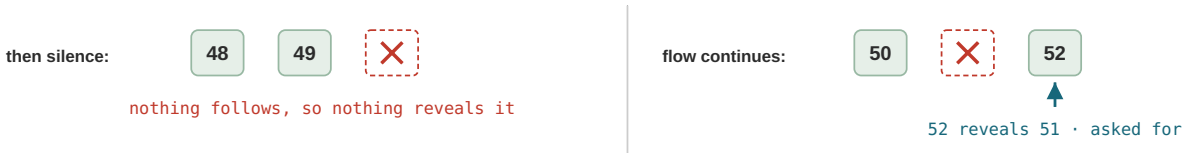

Figure 9 - the blind spot. A lost final frame has no successor to reveal it; a lost frame in a continuing flow is revealed by the next.

11. What the consumer sees

The receiver in every figure above is the fabric's edge, beside the router where a tree ends. The edge runs the whole of this: it tracks flows, asks caches, takes resends, drops duplicates, completes fragments, and only then writes whole objects, in order, to the unicast stream inside each consumer's tunnel, as *Pushed delivery into a Teranode cluster* and *Landing in Kubernetes* specify. The consumer's own path is a TCP connection, which repairs itself the ordinary way. A consumer never sees a gap, a request, a resend or a cache; it sees objects arriving complete.

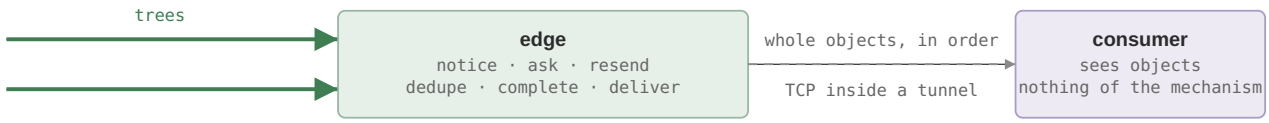

Figure 10 - repair finishes at the edge. The consumer's stream carries whole objects and nothing of the mechanism.

12. Why it scales

Repair is the part of a reliable multicast design that most often fails to scale, because a source that answers its receivers is back to a thousand conversations. The fabric's repair never involves the source. It is keyed to a stamp set once at the door, served by caches that scale with the regions they sit in, bounded by a window so a cache's memory is finite, bounded by rate tiers so a request cannot be amplified, and bounded to one relayed hop so a region's caches cannot pull the whole fabric. A receiver asks only for what it can name, a cache answers only what it holds, and a resend is only ever the frame itself. That is why loss on the Internet is a cost the fabric pays in proportion and not a reason it cannot be built there, which is the claim *Multicast over a unicast Internet* makes and this paper closes.

13. Assumptions and limits

Claim	Basis	Note
Frames carry a flow key and a sequence number from the ingress	**verified**	BRC-124
The request names a flow and a range; three answers	**verified**	BRC-126
Caches announce themselves with distance and preference	**verified**	BRC-126
A relayed request is marked and relayed at most once	**verified**	BRC-126
Fragments are frames with their own numbers	**verified**	BRC-130
A cache keeps frames for a bounded window	**structural**	the window is an operator setting, on the order of a minute, stated
A resend to the requester or into the group	**verified**	BRC-126; the choice is an operator setting
10,000 frames a second, 1500-byte frames, the loss rates	**stated**	the worked example of §9 only
Request size, a few dozen bytes	**verified**	BRC-126
Repair work is proportional to loss	**structural**	one request and one resend per lost frame

Limits, stated plainly:

1. **The last frame of a silent flow is invisible until the flow resumes.** §10 states the exposure and the optional marker that closes it.
2. **A frame older than the cache window cannot be repaired.** The window bounds a cache's memory and therefore the time a receiver has to notice.
3. **The relay is one hop.** Loss that happened before the upstream cache too is not recovered by relaying further; that is a deliberate bound, not an oversight.
4. **A resend into the group reaches receivers that did not ask.** Where delivered bytes are metered, an operator resends to the requester alone; the protocol allows both.
5. **The worked example assumes independent losses.** A burst loss costs one request naming a range and one resend of several frames, which is cheaper per frame, not dearer.

References

BRC-124: Multicast Transaction Frame Format: https://github.com/bsv-blockchain/BRCs/blob/master/transactions/0124.md
BRC-126: Multicast Transaction NACK Retransmission Protocol: https://github.com/bsv-blockchain/BRCs/blob/master/transactions/0126.md
BRC-130: Multicast Transaction Frame Fragmentation: https://github.com/bsv-blockchain/BRCs/blob/master/transactions/0130.md
BRC-139: Multicast Shard Manifest Announcement Protocol: https://github.com/bsv-blockchain/BRCs/blob/master/transactions/0139.md
Pushed delivery into a Teranode cluster (Lightweb Inc., Paper 001): https://1bsv.net/papers/pushed-delivery-teranode.pdf
Landing in Kubernetes (Lightweb Inc., Paper 004): https://1bsv.net/papers/kubernetes-landing.pdf
Multicast over a unicast Internet (Lightweb Inc., Paper 012): https://1bsv.net/papers/multicast-over-unicast-internet.pdf
Frames on a 1500-byte Internet (Lightweb Inc., Paper 013): https://1bsv.net/papers/frames-on-a-1500-byte-internet.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.

© 2026 Lightweb Inc. All rights reserved. 1BSV and Unbounded Solutions for a Small World are trademarks of Lightweb Inc. This paper describes an architecture in rapid development: the design, its interfaces and this document are subject to change without notice, and later versions supersede this one. Nothing in this paper is a commitment to ship any feature, an offer of service, or professional advice.