

04

Live, with nobody holding the switch

A broadcast that runs from the network itself, with no platform able to end it mid-sentence or delete the recording afterwards.

BUILD STATE

Claimed

COMPLEXITY



Vision

Going live means borrowing somebody's switch. A camera is pointed at something happening in a street, or a set is played to a few hundred people, and all of it runs through an account that can be suspended between one sentence and the next. It rarely takes anything dramatic. An automated copyright match hears a record playing in the background and mutes the audio. A region gets blocked. The service has an outage. The rules change on a Tuesday.

The account is not really what gets lost. What goes is the part of a broadcast that only existed while it was live, and then the record of it afterwards. Archives sit on terms set by somebody else: recordings expire after a fixed number of days, a terminated channel takes its back catalogue with it, and years of work can stop being reachable while its author still has every file on a drive at home. The files survive; every link anyone ever shared stops working.

For some broadcasts that interruption is the entire point of them. A protest, an eviction, a queue outside a courthouse: a live record is evidence, and it is most likely to be cut at exactly the moment it starts to matter. Anyone streaming that kind of footage is relying on a company to decline a request, which is a thin thing to rely on.

Swarm changes what a broadcast is. Video becomes content in a shared network rather than a slot on a service, and a viewer follows an address that belongs to a key the broadcaster holds. Nothing sits in the middle of the two of them, which means there is nobody to appeal to and, more usefully, nobody to pressure. When the broadcast ends, the recording is the same object the live stream was, at the same address, still reachable.

Paying for it works unlike a subscription. Storage on Swarm is prepaid and anyone can add to it, so an audience that wants a recording kept can keep it themselves, without the broadcaster's involvement and without anyone's permission. The archive outlives whoever first cared about it.

Architecture

Live video is already close to the right shape. HLS breaks a stream into short segments and a playlist listing them; the player fetches the playlist, then the segments, then the playlist again. Swarm only has to supply two things: somewhere for segments to live, and something that moves to point at the newest ones.

Content addressing (shipped). Each segment is uploaded and gets an address derived from its own bytes. That address is stable and identical for everyone, so a popular segment is already spread across the network by the time the next viewer asks for it, and a segment cannot be substituted for a different one without changing its address.

Feeds and Single Owner Chunks (shipped). A Single Owner Chunk is a network address reserved to one keypair, and a Feed is a series of them at predictable addresses, locatable from an Ethereum address and a topic. The playlist lives in a feed. While the stream runs, the publisher appends a new playlist holding a sliding window of recent segment references, and a viewer resolves the latest feed index to get the current window. When the stream stops, the last playlist written is the finished recording, at the same address the live stream used. Live and archive are one artifact, and the difference between them is only whether the feed is still advancing. Both reference implementations write manifests at an explicit index instead of letting the index be inferred, so a publisher that restarts mid-stream resumes at the right place instead of forking its own stream.

Erasure coding (shipped). Uploads can carry a redundancy level from 0 to 4, set with the `swarm-redundancy-level` header, which adds parity chunks so that content reconstructs even when some of its chunks are unavailable; reconstruction happens during a normal download with no extra work by the reader. Both reference implementations upload segments at level 1. Video needs this more than a file download does, because a chunk that is merely slow stalls a player just as effectively as a chunk that is missing, and parity lets the segment complete without waiting for it.

Postage stamps (shipped). Every write is paid for with a postage stamp, and the batch that pays is prepaid: adding `xBZZ` extends how long it lasts, and if it runs out the content is gone. This is what makes the recording persist with no service underwriting it, and what lets an audience fund an archive directly.

Discovery. A second feed can act as a catalog: finished streams get registered in it, so a viewer can find a broadcast without having been handed a link. It is an ordinary feed, so anyone can keep their own catalog of the streams they care about.

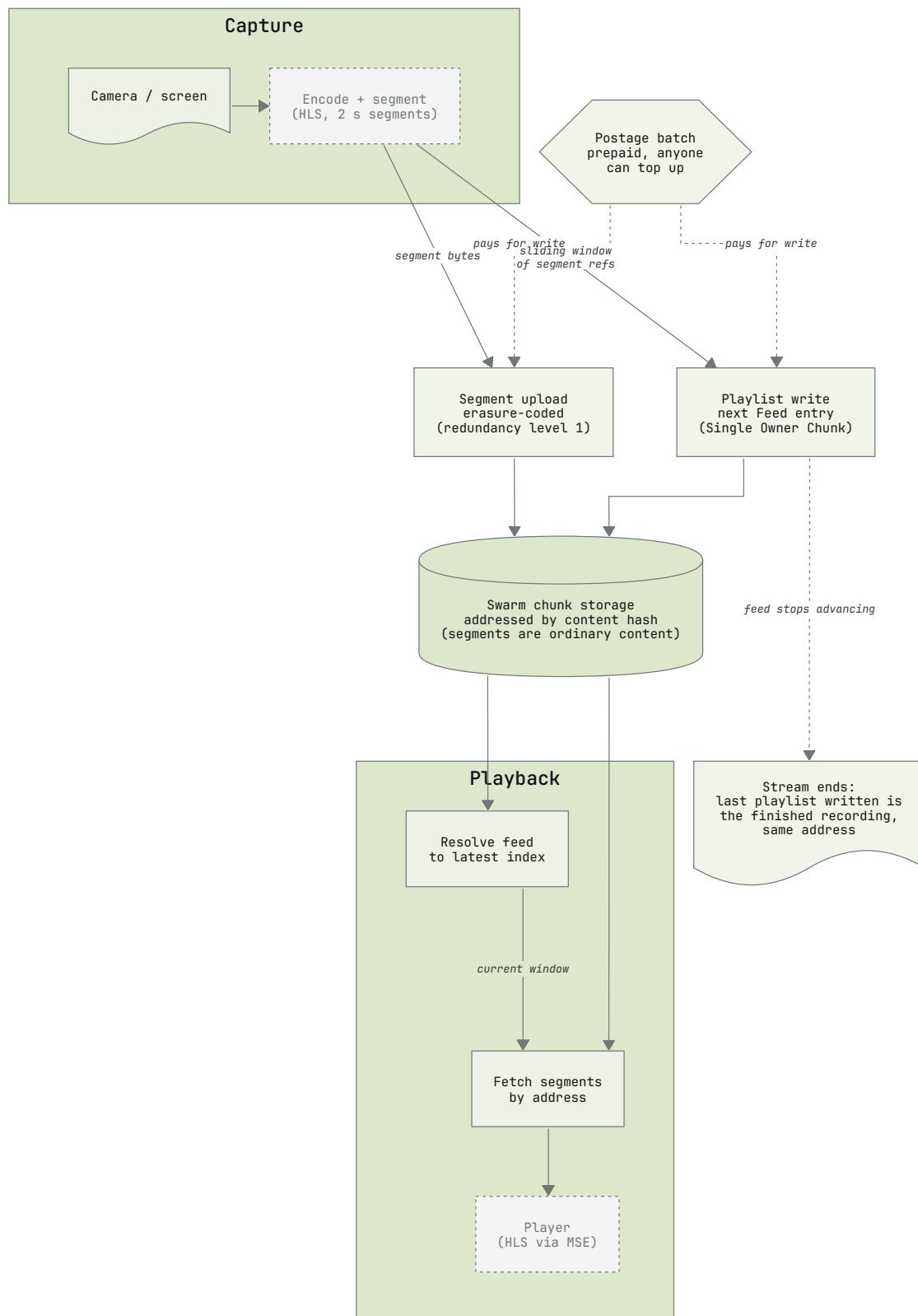
What playback costs today. There are two honest answers. A viewer reading through a Bee node speaks ordinary HTTP to it and gets ordinary player behaviour; running that node themselves leaves nothing in the middle, while using a public gateway trades that back for convenience. A viewer playing entirely inside a browser tab, where a WebAssembly bundle bootstraps an ephemeral light node that speaks the Swarm protocol as soon as the tab loads, with no gateway at all, also works today and has been measured against live mainnet content, but throughput is the constraint: roughly 2 to 7 seconds to retrieve a 1.85 MB segment, around 3 seconds typically, and variable. That is seconds per two seconds of video, so a low-bitrate stream keeps up and a high-bitrate one rebuffers. The limit is how many peers the browser node can reach, not the decoding. Latency has a floor either way, and it is the segment duration: two-second segments put a viewer a few seconds behind live, the same as HLS anywhere else.

PRIMITIVES

Content addressing Single Owner Chunks Feeds Erasure coding Postage stamps

EXTERNAL DEPENDENCIES

HLS WebCodecs hls.js mp4-muxer SRS / Oven Media Engine weeb-3



Data flow. A camera feed is segmented, each segment uploaded as erasure-coded content and each new playlist written as the next Feed entry. Both land in Swarm's chunk storage as ordinary content. A player resolves the feed to the current window and fetches segments by address; when the publisher stops writing, the last playlist is the finished recording at the same address. A postage batch underwrites every write.

Development

There are two ways to publish, and they differ in how much machinery a broadcaster needs rather than in what ends up on Swarm.

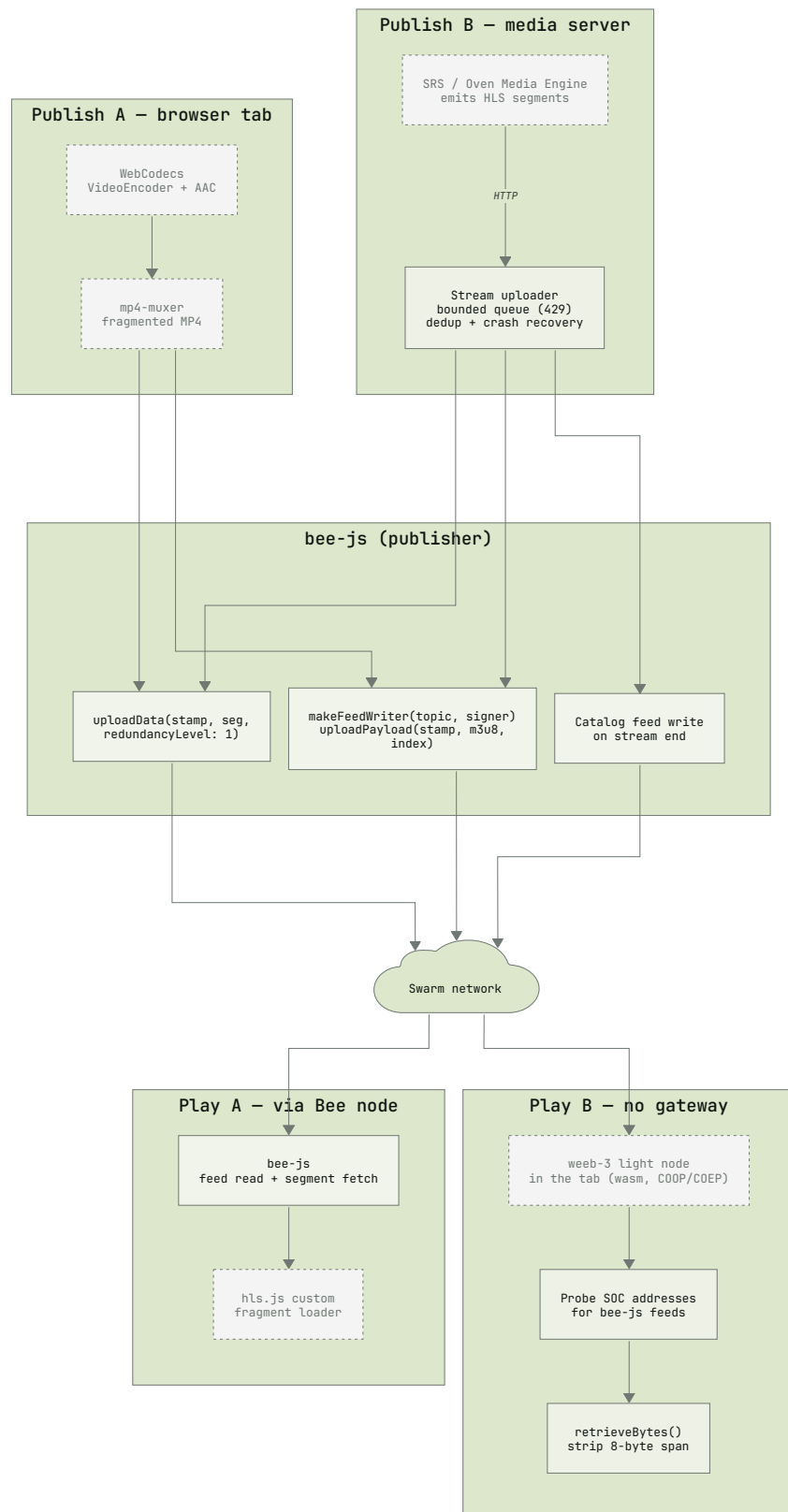
Publish from a browser tab. Capture the camera, encode with the browser's own `VideoEncoder` from **WebCodecs** (*external*) plus AAC-LC audio, mux to fragmented MP4 with **mp4-muxer** (*external*), and upload each fragment with `bee.uploadData(stamp, data, { redundancyLevel: 1 })`. Write the sliding-window playlist with `bee.makeFeedWriter(Topic.fromString(uuid), signer)`. Two-second segments and a six-segment window are a reasonable starting point. AAC rather than Opus, despite Opus having better WebCodecs support, because AAC is what fragmented-MP4 players decode reliably through Media Source Extensions. Generate the feed signer once and keep it in `localStorage` so a reload does not invalidate the link already shared. Pointing `bee-js` at the subsidized `bzz.limo` gateway with a null stamp lets a first broadcast happen before any BZZ has been acquired, which is useful for getting something on screen on day one and not what a real deployment should depend on.

Publish from a media server. Anything that emits HLS segments can feed an uploader service over HTTP; **SRS** and **Oven Media Engine** (*both external*) are the engines already wired up in the reference implementation. The uploader is where the operational work lives, and it is worth copying rather than rediscovering: a bounded per-stream queue that answers 429 when it overflows, so a media server pushing faster than Swarm accepts gets told instead of silently dropped; rejection of duplicate segments; persisted state and a recovery timeout so a crash mid-stream resumes at the right feed index; and registration in the catalog feed when the stream ends. If the media server writes segments to a shared volume, it has to sit on the same host as the uploader.

Play it back. The short path is **hls.js** (*external*) with a custom fragment loader that resolves each playlist entry through `bee-js` instead of fetching a URL; the player does not need to know the difference. For playback with no gateway, boot a **weeb-3** (*external*) light node in the tab, an ephemeral one that comes up with the page and speaks the protocol directly, and pull segments with its `retrieveBytes`, which decodes erasure coding natively and returns an eight-byte span prefix ahead of the content. Two things to expect there: the tab needs WebAssembly and COOP/COEP headers because the node uses `SharedArrayBuffer`, and `weeb-3`'s native feed lookup follows a different feed convention than `bee-js` sequential feeds, so a stream published with `bee-js` is read by probing the Single Owner Chunk addresses directly.

TOOLS & REFERENCES

- `bee-js` library — <https://bee-js.ethswarm.org>
- Bee documentation — <https://docs.ethswarm.org>
- Feeds — <https://docs.ethswarm.org/docs/develop/tools-and-features/feeds/>
- Erasure coding — <https://docs.ethswarm.org/docs/develop/tools-and-features/erasure-coding>
- Postage stamps, top-up and dilution — <https://docs.ethswarm.org/docs/develop/tools-and-features/buy-a-stamp-batch>
- `swarm-hls-stream` — <https://github.com/Solar-Punk-Ltd/swarm-hls-stream>
- In-browser-node playback prototype — <https://github.com/slapec93/poc-in-browser-node>
- Browser-only publisher prototype — <https://github.com/slapec93/ethercast/tree/feat/bzz.limo>
- **External dependencies (not Swarm primitives):** HLS · WebCodecs · `hls.js` · `mp4-muxer` · SRS · Oven Media Engine · `weeb-3`



Component assembly. Two publish paths converge on the same bee-js calls: a browser tab encoding with WebCodecs and muxing to fragmented MP4, or a media server emitting HLS into an uploader service that adds backpressure, deduplication and crash recovery. On the playback side an hls.js fragment loader reads through bee-js, or a weeb-3 light node reads with no gateway at all. External components are drawn dashed to mark them non-Swarm.

Reference implementations

swarm-hls-stream (Solar-Punk-Ltd, MIT) is the complete one, and it is a whitelabel project: a pluggable media-server engine, an uploader with backpressure and crash recovery, a React player with a quality-of-experience overlay, a CLI for node and stamp management, and config-driven deployment across local and remote targets. Live streams are finalized as VOD when they end and registered in a catalog feed. Solar Punk are still working on making the solution more decentralised, so the interesting territory here is not whether streaming over Swarm works but how much of the delivery path can stop depending on a server.

poc-in-browser-node (slapec93) is a prototype, and it is the one that answers the interesting question: playback with no gateway and no node of your own. Verified in Chrome against live mainnet content, including a 646-segment stream. Its README publishes its own throughput measurements instead of a best case, which is where the numbers in the Architecture section come from.

ethercast, branch `feat/bzz.limo` (slapec93) is a prototype broadcaster with no media server at all: capture, encode, mux, upload and feed-write in about three hundred and fifty lines running in a browser tab.