Wallflower.tv

Architecture — Relay-Blind Streaming & the Watch-by-Pubkey Federated Multi-CDN

Version 5 · July 2026

Wallflower streams live video over Media over QUIC (MoQ). Two design commitments shape the system. First, its security model separates *who may reach a relay* from *who may watch the media*, so the servers moving the bytes never see the content (§1–6). Second, streams are named by self-generated cryptographic identities, and this document is explicit about how those names are *routed* to a live relay today versus where that routing is headed (§7).

1. The components

- **Browser** — runs the MoQ publisher (broadcaster) or watcher (viewer), built on the open-source MoQplay engine. This is the *only* place plaintext media ever exists; frames are encrypted before they leave the tab and decrypted only after they arrive in another tab. The browser also mints each stream's cryptographic identity (§7.1).
- `wallflower-iroh` **Worker** (Cloudflare) — the control brain. It authenticates to the relay fleet, mints per-session tokens, and holds the D1 database mapping each broadcast to its relay and content key. It never touches media bytes.
- **Autoscaler / control plane** — `ams.moqcdn.net`, the TinyMoQ fleet manager. The Worker asks it to `/assign` a relay for a broadcast (by name) and to `/release` it on end. It boots or reuses a micro-relay per broadcast and is fail-closed: a missing or wrong credential yields `401`.
- **Relays** — `ams.moqcdn.net:800X`, the data plane. Hermit-OS unikernels under Linux KVM, each serving one broadcast on a dedicated UDP port. Browsers reach them over WebTransport/QUIC; a relay can also pull from another fleet's origin relay over iroh, dialling it by the origin's iroh **EndpointId** (§7.3). On the relay-blind flow they only ever handle **ciphertext**.
- **Public Mainline DHT** — *not a server we run.* The BitTorrent Mainline DHT, used as the discovery plane for watch-by-pubkey (§7.4). The browser publishes and resolves a signed record there directly; there is no directory host in the path.

> **Two ways to get a relay.** Provisioning (getting an origin box, and an edge that pulls from it) runs in one of two modes, both content-blind at the fleet: **direct** — the app calls its own fleet box's `/assign` with that box's `tmq_` bearer; **brokered** — the app calls the tinymoq Worker (`/api/publish` , `/api/edge`) with a `cdn_` customer token, and the Worker brokers a box via `/cdn/assign` . Discovery (§7.4) is separate from provisioning and identical in both modes, so a direct-mode broadcaster and a brokered-mode viewer interoperate through the DHT.

2. Two distinct credential layers

The most important idea in the system: there are two token layers doing unrelated jobs. Confusing them is the single biggest source of error.

Layer 1 — the fleet bearer (tenant identity)

The secret `TINYMOQ_PROVISION_KEY` is Wallflower's *customer identity* with the relay fleet. The Worker sends it as `Authorization: Bearer …` to the fleet's `/assign` and `/release` endpoints. It is long-lived, lives only

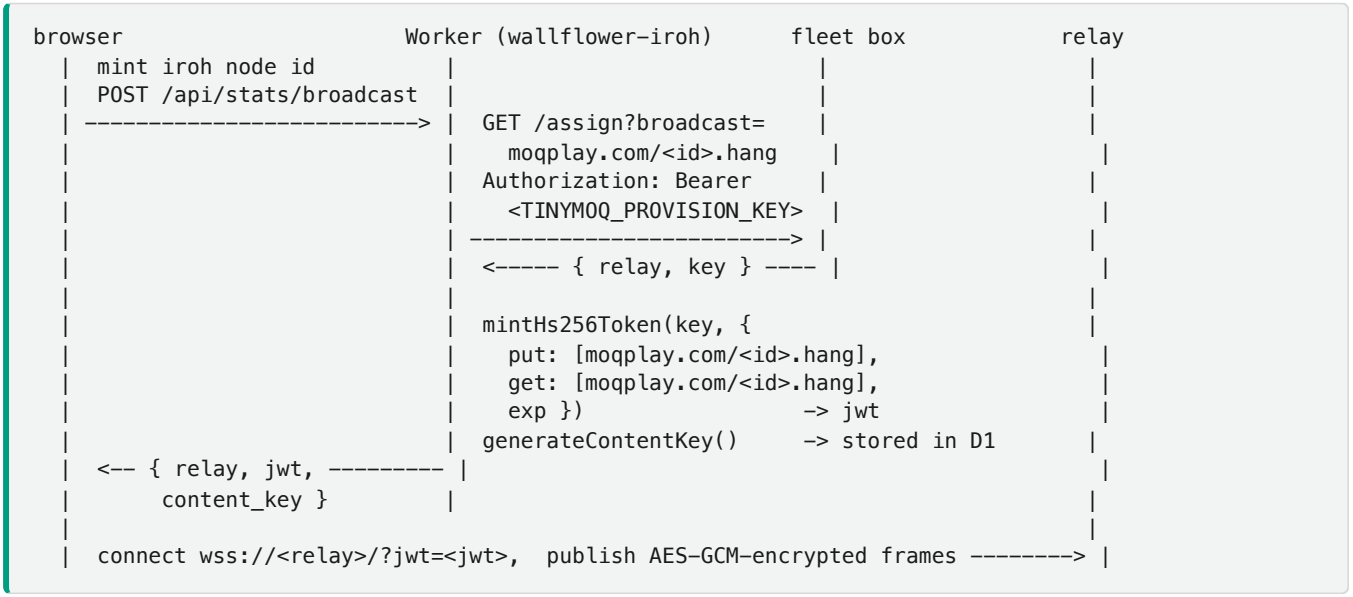
in the Worker's secret store, and **never** reaches a browser or a relay. It proves *Wallflower-the-tenant*, nothing finer-grained.

Layer 2 — per-broadcast MoQ tokens (session capability)

Short-lived HS256 JWTs that authorize *one browser* to do *one thing* on *one relay* for a bounded time. Minted fresh per session by the Worker, couriered to the browser, and presented to the relay in the `?jwt=` connection parameter. Their `put` / `get` claims are scoped to exactly one broadcast name (e.g. `moqplay.com/<id>.hang`), and they are signed with the per-broadcast key the fleet returned from `/assign` — so the relay validates them against its own secret.

> **Key separation.** The bearer proves the *company* to the *fleet*. The MoQ token proves *one browser* to *one relay*. Neither token can decrypt media — that is a third, independent secret (§5).

3. Broadcast flow — going live



1. The browser mints an iroh node id (52-char base32 Ed25519 EndpointId, §7.1) and uses it as the broadcast name, then calls the Worker's `POST /api/stats/broadcast`.
2. The Worker calls the fleet `/assign?broadcast=moqplay.com/<id>.hang` with the bearer. The fleet selects a relay and returns `{ relay, key }`, where `key` is that relay's HS256 signing secret for this broadcast.
3. The Worker mints a JWT (`mintHs256Token`) carrying `put` *and* `get` scoped to the single broadcast name, plus an expiry, signed with that key.
4. The Worker generates a 256-bit AES-GCM **content key** (`generateContentKey`), stores it in D1, and returns it to the broadcaster.
5. The browser connects to `wss://<relay>/?jwt=<jwt>`. The relay validates the JWT against the same `key`; the `put` scope authorizes publishing that name. Frames are AES-GCM encrypted *in the browser* before transmission.

4. Playback flow — joining a stream

```
browser   GET /api/streams/<id>/route --> Worker
                                       |  look up relay in D1 (co-locate on publisher's relay)
                                       |  mintHs256Token(key, { put: [], get: [<name>], exp }) -> v
                                       |  viewerContentKey():
                                       |     public stream   -> return content_key
                                       |     auth-gated       -> return only if signed in
          <-- { relay, jwt, content_key, encrypted } --
          connect wss://<relay>/?jwt=<viewer jwt>,  pull ciphertext, decrypt in browser
```

- The viewer token is `put: []`, `get: [<name>]` — **subscribe-only**. It cannot publish, so a viewer can never hijack the broadcast name.
- The viewer is **geo-routed** to its nearest regional fleet. When that fleet is the origin's own box the broker serves it **directly** (co-location, the fast path); otherwise the broker hands that fleet the origin's `host:port` plus a subscribe-scoped pull token and it **cluster-pulls the origin** over QUIC. Model A media therefore spans regions — co-location is an optimization, not a requirement (live and proven 2026-07-19; §7.2).
- Access control is enforced by *withholding the content key*, not by trusting the relay: an auth-gated stream returns no key unless the caller has a valid session (`viewerContentKey`).

5. The encryption layer is orthogonal to the tokens

MoQ tokens govern *access to the relay*. The **content key** governs *confidentiality of the media*, and it is a wholly separate secret:

- Generated by the Worker, stored in D1, and handed **only** to the publisher and to authorized viewers, always over TLS.
- **Never** given to the fleet control plane or to any relay. A relay authenticates the connection via the MoQ token and forwards the bytes, but those bytes are AES-GCM ciphertext it cannot read.
- This is what *relay-blind* means: a compromised, subpoenaed, or merely curious relay operator sees only encrypted media. Encryption is mandatory — there is no plaintext publishing path.

6. Trust boundaries — what each party can see

Party	Plaintext media	Content key	Fleet bearer	Can publish a name?
Broadcaster browser	**Yes (origin)**	**Yes**	**No**	**Yes (its own)**
Viewer browser (authorized)	**Yes (after decrypt)**	**Yes**	**No**	**No (get-only)**
`wallflower-iroh` Worker	**No**	**Yes (mints/stores)**	**Yes**	n/a (mints tokens)
Relay / fleet	**No (ciphertext only)**	**No**	**No**	Enforces token scope
Network eavesdropper	**No (TLS + AES-GCM)**	**No**	**No**	**No**

7. Stream addressing & routing

A live stream needs a *handle* a viewer can be given, and that handle must resolve to a concrete network location — a relay `host:port`. Wallflower has two ways to do this. Only the first is in the live serving path today; the second is proven and staged for adoption. Conflating them is the most common source of confusion, so this section keeps them strictly separate.

7.1 How a stream is named — and the two identities never to conflate

When a broadcast starts, the browser mints an **Ed25519 keypair** with WebCrypto and derives the **stream key** as `base32(publicKey)` — a 52-character lowercase string. This is the shareable handle and the MoQ broadcast name. It has the *same format* as an iroh EndpointId (both are base32-encoded Ed25519 public keys), but it is **not** a relay's EndpointId — it is the browser's own stream identity. Keeping these two apart is the single most important thing in this section:

	Stream key (identity)	**Origin relay EndpointId (transport)**
What it is	The broadcaster's Ed25519 public key, `base32(publicKey)`	The origin relay's iroh endpoint, from `HERMIT_IROH_SECRET` (64-hex)
Minted by / where	The browser, per broadcast	The relay, random per boot
Role	Names the stream; the capability to watch; the DHT record key	A dialable transport address for cross-fleet origin→edge pulls
Appears in	The share link (`?node=`), the broadcast name	Inside the DHT record's *value* (`origin=<EndpointId>`)

- The handle is *self-generated* and collision-free — no central allocator hands out keys, and there is no URL or port to reserve up front.
- The private half **stays in the tab and is used**: it signs the DHT discovery record (§7.4), so ownership is cryptographically proven — only the key-holder can publish or move the record. (The older fleet-only path minted the key purely as an opaque name; watch-by-pubkey makes the key do real work.)

7.2 Routing model A — broker by name

This is what Wallflower runs in production. The name is resolved to a relay by the TinyMoQ autoscaler — not by the id itself:

```
Worker --GET https://ams.moqcdn.net/assign?broadcast=moqplay.com/<id>.hang--> autoscaler
        Authorization: Bearer <tenant token>
autoscaler -->  { relay: "ams.moqcdn.net:800X", key, byok:false }
                (boots or reuses a micro-relay for this broadcast on UDP 800X)
```

The address handed back is a `host:port`. Viewers do not run discovery themselves: they call the Worker's `/route`, which looks up the broadcast's **origin** relay in D1 and hands the broker that origin plus a subscribe-scoped **pull token**. The broker geo-routes the viewer to its nearest healthy fleet: if that fleet *is* the origin box it serves **direct** (co-location, the fast path); otherwise that fleet **cluster-pulls the origin** over `host:port` QUIC using the pull token. Relays no longer have to be islands — a Model A viewer can be served from a fleet in a different region than the origin, the media pulled cross-fleet (live 2026-07-19; §7.5 covers the pull-leg auth).

Viewer steering. `/route` places a viewer three ways: **Mode A (origin)** — served directly on the origin's box when the viewer is nearest it; **Mode B (cross-fleet edge)** — the viewer's geo-nearest fleet pulls from the origin, authorized by a *subscribe-scoped* pull token (the same scope as the viewer token, with no cluster flag — the relay authorizes the pull by scope alone); **Mode C (enterprise on-net)** — a private relay on the viewer's own network (ASN-gated), so media never traverses a public server. Under the brokered production path, Modes A and B are the broker's **automatic** choice per viewer geo — no override needed — and Mode B is **live and proven** (2026-07-19: a viewer geo-forced to Amsterdam was served by the ams fleet cluster-pulling a Washington DC origin over `host:port`). The broker owns co-location detection by hostname, so it never hair-pins. (This was carried on the `/cdn/assign` contract all along, but the brokered `/route` previously sent only `{broadcast}` — dropping origin and pull — so auto-geo Mode B did not actually pull until the Worker was fixed to forward them.) In every mode the content key is delivered by the same rule (§5) and the relay-blind property holds.

7.3 The relay fleet

The data plane is a fleet of **micro-relays**. The autoscaler boots each on demand as a **Hermit-OS unikernel** — a single-address-space Rust image running under **Linux KVM** — listening on a dedicated UDP port. Each relay is per-broadcast keyed and short-lived; idle relays are reaped and their `/release`d slots recycled. The browser↔relay hop is always WebTransport/QUIC to a `host:port`. For cross-fleet federation, a relay also carries an **iroh endpoint** of its own (derived from its `HERMIT_IROH_SECRET`): an edge relay dials the origin relay by that **EndpointId** over iroh — mainline-DHT discovery, NAT-friendly, so an origin needs no public `host:port`. (Where both origin and edge have routable addresses, native `moq-relay` cluster-connect by `host:port` is still available; the `?xport=iroh` selector picks between them, §7.5.)

7.4 Routing model B — watch by pubkey, resolved off the DHT

The watch-by-pubkey path replaces the central broker with *a self-signed record on a public DHT*. There is **no directory server** — discovery runs entirely in the browser. It is live today on the `?node=` watch path.

```
1. Publish.  broadcaster's browser signs a pkarr (BEP44) record with its PRIVATE key
   and PUTs it to the public Mainline DHT, keyed by z32(publicKey):
       value = "origin=<origin-relay EndpointId>;name=<base32 stream key>;access=public"
   (publishing presents a put-scoped token {put:[key]} the Worker mints in
    /api/publish — the relay fleet is now token-gated; see the §7.5 callout)
2. Resolve.  viewer is given the stream key out of band (a share link), then the
   viewer's browser reads that key's record straight off the Mainline DHT.
3. Provision + watch.  the viewer's fleet assigns an edge that PULLS from the origin
   relay over iroh (dialling the origin EndpointId from the record); the browser
   watches from that edge over WebTransport, presenting a subscribe-scoped token
   {get:[key]} the Worker mints — the relay is token-gated (the key stays the
   shareable capability: holding it is what authorizes the mint).
```

The mental shift: there is no location to look up in a database and no server to ask — the **identity is the public key**, and the key both names the stream and, being 256-bit and unguessable, *is* the capability to watch ("knows the key = may watch"). The record is **self-authenticating**: the DHT accepts a write under a key only if it is signed by that key, so nobody but the broadcaster can publish or move a stream's record — no registration authority required. A stream keeps the same key even if it migrates to a different origin relay; the browser simply re-publishes the record with a new `origin=`.

7.5 The DHT record and the origin→edge pull

The record's *value* carries the origin relay's iroh **EndpointId** (§7.1) — the transport address the record author does not choose but reads back from provisioning. An edge resolves the stream key to that EndpointId and pulls from the origin over **iroh** (mainline-DHT discovery, NAT-friendly). Where the origin also has a routable `host:port`, native `moq-relay` cluster-connect over QUIC is the alternative; the viewer's `?xport=iroh` selector picks between them. In both cases the browser↔edge hop is always WebTransport

— iroh only ever appears on the origin→edge hop, never at the browser. **Pull-leg auth (2026-07):** now that origins are token-gated, the edge authenticates its pull by presenting the viewer's subscribe token over the cluster-connect. That token is conveyed to the origin only over the `host:port` QUIC connect (the URL query reaches the origin's auth), so live cross-box pulls currently use `host:port`; conveying the token over the *iroh* transport is a known, deferred fix — needed only for a NAT'd origin, and the current fleet is publicly reachable. The **brokered MoQplay (Model A)** cross-fleet pull now uses this exact mechanism — a geo-nearest edge presents the viewer's subscribe-scoped token to the gated origin over the `host:port` cluster-connect — verified 2026-07-19: the origin authorized the pull on the real broadcast scope (`internal=false`, and no anonymous sentinel anywhere on the fleet), so both live watch paths share a single pull-leg auth model.

Proven end-to-end on 2026-07-12 (and again in the browser in July 2026 on the current path): a broadcast's stream key was published to the Mainline DHT, resolved by a *different owner's* fleet, which spawned a Hermit edge that pulled from the resolved origin and delivered **real fmp4 media** across two independent fleets (`ams.moqcdn.net` and `was.moqcdn.net`). The stream key is the stable handle; the churning origin transport address is what the record keeps current — which is how it solves the "dynamic origin" problem and dispenses with any centralized lookup.

> **Encryption scope — be precise.** Since 2026-07 this watch-by-pubkey path is **token-gated at the relay**: the Worker mints a per-broadcast subscribe token and token-less connects are rejected. The (unguessable) key remains the shareable capability — holding it is what authorizes the Worker to mint — so access is still "knows the key = may watch," now *enforced* by that token rather than an open relay. Gating adds *transport access-control*, not media crypto: the media on this path is still **not** end-to-end encrypted. The relay-blind, content-key-gated end-to-end encryption of §5 is the separate **MoQplay** flow (§7.2, Model A). Wherever "relays move only ciphertext" is claimed, it means that flow — not this one.

7.6 Two flows, side by side

Both models are live. They are *different products*, not stages of one migration: the MoQplay flow is the encrypted experience with customer-policy tokens; watch-by-pubkey is the decentralized-discovery experience keyed by a shareable pubkey. Both are now **token-gated at the relay** (a per-broadcast token is required on every connect — gating is *per-tenant* and default-off, and the wallflower tenant is gated across the whole fleet); what differs is the *minting policy* — customer-defined vs. auto-issued to any holder of the key — and whether the media is end-to-end encrypted. They share the same relay fleet.

	MoQplay flow (Model A)	Watch-by-pubkey (Model B)
Stream handle	Browser-minted key, used as an opaque broadcast name	Browser-minted key, used as the identity *and* the capability
Discovery	Broadcast name → Worker `/route` → `/assign` broker (D1-backed)	Public key → signed record on the Mainline DHT, resolved in-browser (no server)
Origin→edge transport	Native cluster-connect by `host:port` (or iroh, `?xport=iroh`)	Iroh by the origin's EndpointId from the record (or `host:port`)
Ownership proof	**None (name is opaque)**	**Self-authenticating — the DHT write is signed by the key**
Watch auth	Worker-minted subscribe token (customer policy)	Worker-minted subscribe token (`get:[key]`), auto-issued to any holder of the unguessable key
Media encryption	**Relay-blind end-to-end (content key, §5)**	**None on this path (transport-gated, not media-encrypted)**
Proven on	`ams.moqcdn.net` (production)	Two independent Hermit fleets, browser-to-browser (2026-07)

What retired: the standalone `luke.moqcdn.net` directory and its `/api/register` / `/api/resolve` service. Discovery moved *into the browser* (signed pkarr records on the public Mainline DHT), and provisioning moved to the tinymoq Worker (brokered) or direct-to-box (§1). No node-id directory host remains in the path.

8. Teardown

On stop or tab close, the browser calls `POST /api/stats/broadcast/:id/end`. The Worker marks the broadcast ended in D1 and calls the fleet `/release?broadcast=<name>` (`releaseRelay`), freeing the relay slot. Content keys are per-session; a restarted broadcast receives a fresh key, so previously distributed keys cannot decrypt a new session.

> ### The model in one line
>
> The bearer proves **Wallflower** to the fleet; the fleet returns a per-stream signing key; the Worker uses that key to mint short-lived, tightly-scoped JWTs that prove **this browser** to **this relay**; and a separate content key — shown to no server — keeps the video private end-to-end.

Part II — The federated node-ID multi-CDN

The architecture Wallflower is being built toward.

Part I described the system as it runs today: one tenant, one fleet, broker-by-name routing. This part documents the target — a federated, node-id-addressed content network where many independent operators pool into one global footprint. It reuses every security primitive of Part I; what changes is how streams are *found* and how fleets *interconnect*.

> *iroh-steered federated multi-CDN — scalable, decentralized, and secure, with each property living in its own plane.*

9. Thesis — iroh moves *up* the stack (and covers the origin hop)

The organizing idea: **iroh is first the identity and discovery plane, and second an optional NAT-friendly transport for one hop only — origin→edge.** It never touches the browser: the browser↔edge media hop is always WebTransport (measured ~600+ Mbit/s). Where iroh earns its place is naming and locating streams, and — because a Hermit origin relay carries its own iroh EndpointId (§7.3) — letting an edge dial that origin *by key* so an origin behind NAT needs no public `host:port`. Where the origin is routable, native `moq-relay` cluster-connect by `host:port` is the equivalent path; the two are interchangeable and the viewer's `?xport=iroh` selects.

> ***Where iroh lives — and where it doesn't. Used for identity + discovery:*** *the key format, and a self-signed `pkarr` record on the public Mainline DHT mapping stream key → origin EndpointId.* ***Used as transport, origin→edge only:*** *an edge dials the origin relay's EndpointId over iroh (or falls back to `host:port` cluster-connect).* ***Never used at the browser:*** *browser↔edge is WebTransport, always — iroh is not a browser transport.*

10. Three transport hops, strictly separated

Hop	Transport	Role
browser ↔ relay	**WebTransport, always**	The media path. ~600+ Mbit/s measured. Browsers never speak iroh.
origin ↔ edge (relay↔relay)	**iroh by EndpointId, or QUIC by `host:port`**	Cross-fleet origin→edge media pull. The edge dials the origin relay's iroh EndpointId (NAT-friendly), or native cluster-connect where the origin is routable. Selected by `?xport=`.
discovery	**signed pkarr record on the Mainline DHT**	Naming and resolution, published and read *by the browser* — no directory server, no control-plane hop for discovery. Provisioning (§1) is a separate concern.

A stream's handle is its **stream key** — `base32(ed25519 public key)`, ~52 chars, minted in the browser. It replaces the URL. Unlike a bare locator it is *both identity and capability*: it names the stream, and being 256-bit and unguessable it is the right to watch (§7.4). Resolving its DHT record yields the origin relay's iroh EndpointId — a *separate* key that is the transport address, not the stream name (§7.1). (iroh 1.0 renamed `NodeId` → `EndpointId`; the terms are used interchangeably for the relay's transport key.)

11. The four-plane model

The spine of the architecture. Each of "scalable, decentralized, secure" lives in a *different* plane, so they stop competing.

Plane	Answers	Owner	Stance
Identity / Discovery	"Where is stream key X?"	Public Mainline DHT (browser-published, self-signed)	**Open network, no server we run**
Control / Routing (provisioning)	"Give me an origin box, and an edge that pulls it."	tinymoq Worker (brokered) or fleet box (direct)	Gated — customer token / box bearer
Data	Move the bytes	Relays (relay↔relay native cluster-connect by `host:port`; browser↔relay WT)	—
Content security	"Who may watch? Who may decode?"	The customer (broadcaster)	Gated — customer's choice, per stream

Decentralized *findability* is an open DHT; brokered *routing policy* is a central `cdnadmin`; self-sovereign *access* + *confidentiality* are customer-held tokens and keys. Separating them is what lets the discovery plane be fully open without weakening security (§12).

12. Three security concerns, kept separate

Part I's mechanisms map exactly onto three concerns that must never be conflated. **A** is §2 Layer 1; **B** is §2 Layer 2; **C** is §5.

A — Customer ↔ fleet infrastructure trust

The customer's provisioning credential to the fleet manager. (Wallflower: `TINYMOQ_PROVISION_KEY`.) A separate concern from stream access; not the focus of the federation model.

B — Access control

Two forms, matching the two flows (§7.6):

- **Token-gated (MoQplay).** The **customer mints** broadcast and view tokens; the fleet **enforces** them. A gated stream requires a minted, scoped token; access is enforced *by the token, not by network topology*, so discovering the relay buys nothing without one.
- **Key-as-capability (watch-by-pubkey).** The relay fleet is token-gated, but the stream key *is* the capability: 256-bit and unguessable, "knows the key = may watch." The Worker **auto-mints** a per-broadcast token scoped to the bare stream key for any holder of it — publish (`put:[key]`) and watch (`get:[key]`) — so the viewer never handles a token; sharing the key is what authorizes the mint. Access is thus still controlled by **how widely the key is shared** (out-of-band, like a secret link), now *enforced* by that auto-minted token rather than an open relay. The DHT record is self-authenticating, so only the key-holder can publish or move it (§7.4).

(Token layer today: Worker-minted HS256 tokens keyed to the fleet-returned per-broadcast key. The customer-sovereign form is BYOK EdDSA — `mintEd25519Token` signs with the tenant's private key. Its public half, the `verify_jwk`, is registered with the fleet and gates the *brokered provisioning* path — it authenticates the customer to the Worker/broker; it is *separate from media confidentiality* (concern C) and from the key-as-capability model above.)

C — Confidentiality (media encryption)

The broadcaster **encrypts the media at the source, before it enters QUIC, with a key the broadcaster controls** (end-to-end, relay-blind). Even an operator running a maliciously recompiled relay that forks the stream to disk captures only ciphertext. This is precisely what makes it *safe for a stream to transit other customers' or other fleets' relays* in the federated graph (§13, Diagram 2). B says who is *allowed*; C guarantees who can *actually decode*. In the shared graph you want both. (Wallflower today: Worker-generated AES-GCM content key, handed only to the publisher and authorized viewers, never to any relay.)

13. Two reference topologies

Diagram 1 — one customer, one fleet (the vertical slice)

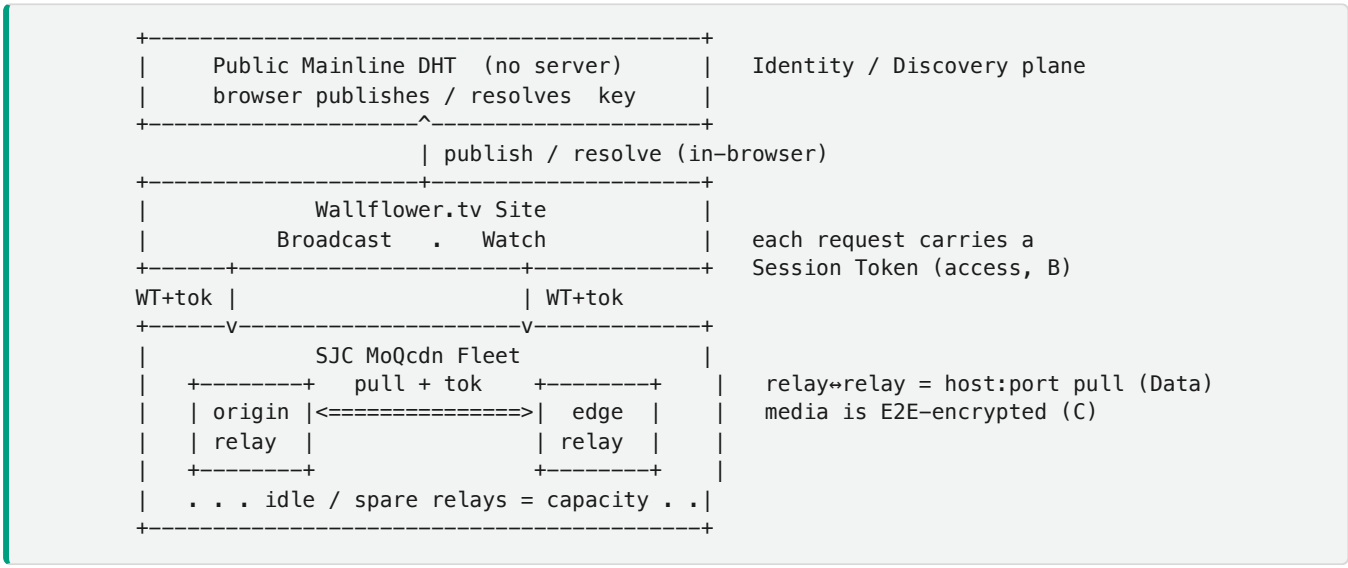


Diagram 2 — two customers, two regions (the interconnect / partner network)

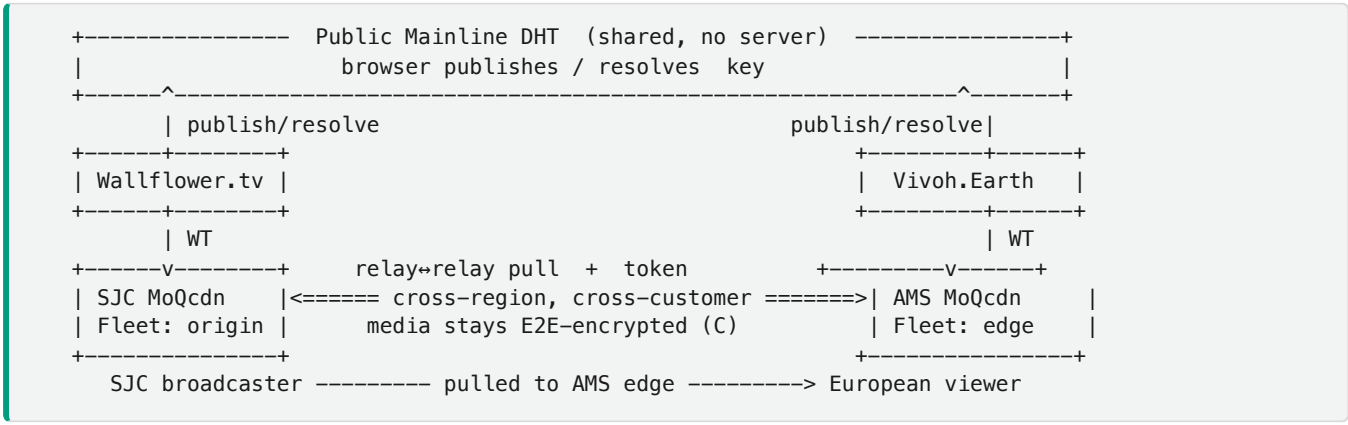


Diagram 2 is the payoff: SJC's relay and AMS's relay federate directly — *across regions and across customers*. A broadcaster whose origin is in SJC can be pulled to a European edge and watched by a nearby viewer. N independent customers become **one pooled global footprint**: each contributes a regional fleet, and any partner's viewer reaches any partner's broadcaster via the nearest edge. This is CDN interconnect / peering, expressed as a shared content graph.

14. NAT-crossing origins — "private islands"

The iroh origin→edge key-dial is a proven mechanism (§7.5, §9), not a future one. The DHT record's `origin=` value is the origin relay's iroh EndpointId, and an edge can dial it over iroh (outbound connect + hole-punch) rather than by `host:port`. Because relay↔relay dials by key, a fleet can in principle be an **origin with zero inbound ports** — no port-forwarding, no public IP — behind home or corporate NAT, and edges still reach it. **One current limit (2026-07):** now that origins are token-gated, the edge's pull must present a subscribe token, and that token reaches the origin's auth only over the `host:port` connect — not yet over the iroh transport — so the live gated fleet pulls cross-box over `host:port`. The iroh-transport

token-conveyance fix (and the *fully self-hosted* iroh-relay + DHT variant below) is what remains forward-looking; the key-dialling itself is proven.

- **Precision:** this is "*no public ingress,*" not "*air-gapped.*" A true air gap cannot federate — you need outbound reachability to a discovery service plus a rendezvous relay to coordinate hole-punching. The mental model is "a fortress with an outbound-only door."
- **Fallback:** if both ends are behind hard/symmetric NAT and hole-punching fails, traffic transits an iroh relay (a performance hit) — and that relay still sees only ciphertext (concern C). You can *self-host* that iroh relay to keep even the fallback inside your domain.
- **Config, not rebuild:** run your own iroh relay and your own DHT for a fully *private island* that peers only with chosen partners, or point at n0's public infrastructure to join the *open* network. "Works with islands or with public networks" is literally true — it is a configuration choice.

15. The cross-partner shared graph — and one open question

The value of the partner network is scale, geographic reach, and independence: partners participate in the DHT and volunteer to act as **edges** for another partner's audience, forming a shared content graph in which one customer's streams are discoverable and watchable across the network (subject to the origin's **access policy** — for a public stream, simply knowing the pubkey; for a gated stream, a minted token). **Geo-nearest fleet routing is now live** (2026-07): the broker ranks the customer's healthy fleets by great-circle distance from the viewer — whose location the app forwards from the browser (`request.cf`) — and assigns the nearest, proven across four regional fleets (Amsterdam / Tokyo / Washington DC / Santa Clara), with a spill to the next-nearest when a box is full or unreachable. **As of 2026-07-19 this completes the media path end-to-end:** when the geo-nearest fleet is not the origin, the broker hands it the origin `host:port` plus a subscribe-scoped pull token and that fleet cluster-pulls the origin over QUIC — proven Amsterdam↔Washington DC (stream `lq0ct`) — so a viewer is served from a nearby fleet even when the broadcaster is in another region. This is the same origin+pull mechanism a delegated *partner* fleet will use to serve another customer's viewers. What remains out of scope is DNS-layer request-routing (anycast / geoDNS). **Metering** is further along: the boxes already emit per-box, per-customer usage (`/metering`, `relay_seconds`), so per-customer consumption is measured today — what remains open is **cross-partner attribution** (whose egress when one partner's edge serves another's viewer) and the **settlement** policy layered on top. That cross-org accounting is a real interconnect concern — it is why CDN peering has settlement — and is left here as a future control-plane matter.

Control-plane precedent. `tinymoq.com/cdnadmin` already has fleets registering up to it; the `cdnadmin` console holds the customer registry and fleet delegation. The planned handoff: `cdnadmin` routes a request to a downstream fleet and passes the origin's **iroh EndpointId** plus the **pull token** the spawned edge will present to that origin. (Keep that pull token distinct from the **provisioning bearer** — `cdn_` / `tmq_`, control-plane, Worker-held — that the fleet uses to spawn the edge in the first place; the callout below is about the pull/view token, not the bearer.) The iroh change is simply that the passed origin **becomes an iroh EndpointId instead of a `host:port`**, so an edge can pull from an origin that has no public address (§14). (These are two *different* identities and stay separate throughout: the broadcaster's browser pubkey names the stream, §7.1; the origin relay's EndpointId is the transport endpoint here.)

> **Open question — the cross-org handoff, in two halves.** When a Vivoh viewer watches a Wallflower stream, content belongs to the origin customer and access is customer-owned, so the **origin (Wallflower) decides**

access even though the viewer arrived via Vivoh's edge. The question splits exactly along the B/C decomposition (§12):

- **(B) Access.** For a **public stream the origin auto-mints the token locally** — any holder of the pubkey is issued a scoped publish/subscribe token by the origin's own Worker, so knowing the pubkey *is* the capability (the DHT record is self-authenticating) and no *cross-customer* token negotiation is needed. Only a **policy-gated stream** (the customer restricts who may watch) needs a cross-customer token, which `cdnadmin` brokers (the viewer's site asks, the origin's policy answers). That cross-org gated path is heavier and is deferred.
- **(C) Confidentiality.** The harder half. An authorized cross-partner viewer handed a valid token still holds only *ciphertext* — they also need the origin's decryption **key**, and key distribution across trust domains is the genuinely hard part of any E2E system. What makes C harder than B: a **token** can be brokered then discarded (short-lived, scoped, revocable), but a **key** cannot — possession is permanent, so revoking access means *rekeying the stream*, not tearing up a token. C's cross-org handoff therefore needs forward-secrecy / key-rotation thinking that B does not.

Today the shipping system **keeps the two flows separate**: the public pubkey-federation watch path is deliberately *not* E2E (the pubkey is the capability), while the relay-blind ciphertext path (MoQplay) *is* E2E but is not part of the open graph. **Public streams dissolve the problem**; the interesting case is **gated cross-org E2E** — unifying the two flows — which the two-plane resolution below shows is solved in shape, and staged in delivery.

15.1 Confidentiality (concern C) — the two-plane resolution

Access (B) governs who may watch; confidentiality (C) governs who can decode. They look entangled only because it is tempting to distribute keys along the same path the media takes — but they need not share a path, and separating them dissolves the apparent conflict between end-to-end encryption and the open graph.

Split the system into two orthogonal planes. The **media plane** is content-blind: the origin encrypts each frame and relays/edges — any partner's — fan out opaque ciphertext, exactly as the relay-blind flow already does. The **key plane** is a thin, origin-controlled channel that delivers the content key to authorized viewers; it rides the control plane (`cdnadmin`'s brokered handoff), never the media fabric, and is opaque to every partner and to the CDN itself. With this split, both of the graph's promises hold at once: any partner's viewer watches any partner's stream (the ciphertext reaches them over the shared graph) *and* only key-holders decode (the key reaches them only over the origin's key plane). Partners move bytes; the origin alone decides who decodes; the edge is a dumb ciphertext pipe.

The mechanism. Media is encrypted per-frame with **SFrame** (RFC 9605), designed for exactly this — E2E media over untrusted relays; the relay sees ciphertext and headers, never plaintext. The frame key derives from an epoch secret (`CK_epoch = HKDF(groupSecret, epoch)`) that rotates on a timer or on any membership change. Delivery of `CK_epoch` to a viewer takes one of two forms, staged by need:

- **Near term — envelope wrapping (HPKE, RFC 9180).** Each viewer already mints a browser keypair for the pubkey graph; it gains an encryption key alongside its signing key. To admit a viewer, the origin (or a keyserver it delegates) HPKE-wraps `CK_epoch` to that viewer's public key and returns the wrapped blob over `cdnadmin`'s brokered channel — opaque to `cdnadmin` and to every edge, unwrappable only by the viewer. No group state; works across trust domains today.
- **At scale — MLS (RFC 9420).** A continuous group-key-agreement protocol for the authorized-viewer set, with forward secrecy, post-compromise security, and log(N) add/remove. The authorized-viewer set is the group, the origin is its administrator, and `cdnadmin` serves as the MLS Delivery Service — ordering handshake messages without ever being a trust anchor or seeing a key. MLS exports the

secret the SFrame epoch derives from; this is the same MLS-over-SFrame construction the IETF is standardizing for end-to-end media.

Access binds to confidentiality at one clean join. The origin-minted view token is scoped to the viewer's public key, so the rule is auditable: *wrap* `CK` *only to a key that holds a valid token.* B decides who is wrapped; C delivers the key. That dissolves the two properties that made C harder than B. A key's permanence stops mattering, because revocation is no longer “rekey and redistribute” but simply *do not re-wrap the next epoch to the revoked viewer* — they retain only what they could already decode, and MLS makes even that step cheap while healing the group. And the forward-secrecy / key-rotation reasoning C demands is supplied off the shelf by the epoch ratchet or the MLS tree, not invented here.

What this does not buy — stated plainly. End-to-end encryption here means a content-blind *fabric*, not unexfiltratable content: it prevents relays and partners from seeing plaintext, not an authorized viewer from re-recording what they are entitled to watch. It does not hide **metadata** — stream existence, who-watches-what, and timing/size patterns remain visible to the graph. And it *relocates*, rather than removes, a trust-and-availability anchor: someone must still assert “this viewer is admitted,” but that someone is the origin (or its delegate), never a partner or the CDN.

> **Status: solved in shape, staged in delivery.** Every primitive is a standard (SFrame, HPKE, MLS), and the load-bearing pieces already run in this system — browser-minted keys, relay-blind media, and `cdnadmin` 's brokered handoff. None of the C-specific wiring (SFrame framing, the per-viewer encryption key, the key-wrap endpoint) is built yet. What remains genuinely open is not the cryptography but the *policy*: the cross-organization membership and delegation model (who may grant admit rights on whose behalf) and the limits of metadata privacy on an open graph. Those are control-plane and governance questions — narrower, and more tractable, than “end-to-end encryption across the graph is unsolved.”

16. Signed records — why open discovery is trustworthy

A public, server-less discovery plane invites an obvious worry: if anyone can write, what stops key→origin poisoning, squatting, or spam? The pkarr/BEP44 design answers it structurally, and Wallflower now relies on that answer rather than a registration authority:

- **Writes are self-authenticating.** A Mainline-DHT mutable record lives under `z32(publicKey)` and the DHT accepts a PUT only if it is signed by that key. So the only party who can publish or move a stream's record is the holder of the stream key's private half — which is the broadcaster's browser, and no one else. There is nothing to squat: a record under key *K* can only be written by *K*.
- **The precise rule holds without a server:** *resolution is open; writes are authenticated by the key.* No `/api/register`, no directory operator to trust, no allow-list — the signature *is* the authorization. This is what retired the standalone directory (§7.6): the property it was going to provide is a native property of the DHT.