Wallflower.tv

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

Version 4 · July 2026 · Worker: `wallflower-iroh` · Relay fleet: `ams.gpcmoq.com` / `dc1.moqcdn.net` (TinyMoQ)

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). Where a mechanism is in production it is marked LIVE; where it is demonstrated but not yet in the serving path, PROVEN. **Part I (§1–8)** is the system as it runs today; **Part II (§9–17)** documents the federated node-id multi-CDN it is being built toward, with an explicit proven-vs-not-built status (§17).

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.gpcmoq.com`, 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.gpcmoq.com: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

```
browser                        Worker (wallflower-iroh)      fleet box             relay
  |  mint iroh node id            |                             |                     |
  |  POST /api/stats/broadcast    |                             |                     |
  | --------------------------->  |  GET /assign?broadcast=     |                     |
  |                               |    moqplay.com/<id>.hang    |                     |
  |                               |  Authorization: Bearer      |                     |
  |                               |    <TINYMOQ_PROVISION_KEY>  |                     |
  |                               | -------------------------> |                     |
  |                               |  <----- { relay, key } ---- |                     |
  |                               |                             |                     |
  |                               |  mintHs256Token(key, {                            |
  |                               |    put: [moqplay.com/<id>.hang],                  |
  |                               |    get: [moqplay.com/<id>.hang],                  |
  |                               |    exp })                -> jwt                   |
  |                               |  generateContentKey()    -> stored in D1          |
  |  <-- { relay, jwt, ---------- |                                                   |
  |       content_key }           |                                                   |
  |                                                                                   |
  |  connect wss://<relay>/?jwt=<jwt>,  publish AES-GCM-encrypted frames --------->   |
```

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 }) -> vi
                                         |  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 steered to the **same relay** as the publisher. Relays are islands (they do not federate by default), so co-location is required for the media to be reachable.
- 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 LIVE

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.gpcmoq.com/assign?broadcast=moqplay.com/<id>.hang--> autoscaler
        Authorization: Bearer <tenant token>
autoscaler -->  { relay: "ams.gpcmoq.com: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 current relay in D1 and (idempotently) re-asks `/assign` so the viewer **co-locates** on the same relay as the publisher. Relays are islands — they do not federate by default — so co-location is mandatory for the media to be reachable.

Viewer steering. `/route` can place a viewer three ways: **Mode A (origin)** — the publisher's own relay (default); **Mode B (cross-cluster edge)** — a nearer relay that pulls from the origin, authorized by an extra cluster-scoped pull token; **Mode C**

(enterprise on-net) — a private relay on the viewer's own network (ASN-gated), so media never traverses a public server. 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 LIVE

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 to the relay itself is tokenless — the relay is open-publish)
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 (open subscribe — the key is the capability).
```

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 PROVEN

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.

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 `ams.gpcmoq.com`). 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.** This watch-by-pubkey path rides the relay's *open subscribe*: access is gated only by knowing the (unguessable) key, and the media on this path is **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, token-gated experience; watch-by-pubkey is the open, decentralized-discovery experience. 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	Open subscribe — the unguessable key is the capability
Media encryption	**Relay-blind end-to-end (content key, §5)**	**None on this path (open subscribe)**
Proven on	`ams.gpcmoq.com` (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. Proven primitives and unbuilt work are labelled throughout; §17 is the status of record.

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 runs open subscribe (`moq-relay --auth-public-subscribe ""`) and the stream key *is* the capability: 256-bit and unguessable, "knows the key = may watch." Access is controlled by **how widely the key is shared** (out-of-band, like a secret link), not by a token. Publishing is likewise tokenless — the relay is open-publish and the DHT record is self-authenticating (§7.4).

(Token layer today: Worker-minted HS256 tokens keyed to the fleet-returned per-broadcast key LIVE. 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 **LIVE**.)

13. Two reference topologies

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

```
          +--------------------------------------------+
          |     Public Mainline DHT  (no server)      |   Identity / Discovery plane
          |     browser publishes / resolves  key     |
          +---------------------^----------------------+
                                | publish / resolve (in-browser)
          +---------------------+----------------------+
          |             Wallflower.tv Site             |
          |          Broadcast   .   Watch             |   each request carries a
          +------+----------------------+--------------+   Session Token (access, B)
          WT+tok |                      | WT+tok
          +------v----------------------v--------------+
          |             SJC MoQcdn Fleet               |
          |   +---------+    pull + tok    +---------+    |    relay↔relay = host:port pull (Data)
          |   | origin |<================>|  edge  |    |    media is E2E-encrypted (C)
          |   | relay  |                  | relay  |    |
          |   +---------+                  +---------+    |
          |   . . . idle / spare relays = capacity . .|
          +--------------------------------------------+
```

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

```
     +----------------  Public Mainline DHT  (shared, no server)  ----------------+
     |                  browser publishes / resolves  key                         |
     +------^------------------------------------------------------------^-------+
            | publish/resolve                                 publish/resolve|
     +------+---------+                                     +----------+-------+
     | Wallflower.tv |                                     |  Vivoh.Earth   |
     +------+---------+                                     +----------+-------+
            | WT                                                       | WT
     +------v---------+     relay↔relay pull  +  token         +---------v------+
     | SJC MoQcdn    |<====== cross-region, cross-customer =======>| AMS MoQcdn     |
     | Fleet: origin |      media stays E2E-encrypted (C)          | Fleet: edge    |
     +----------------+                                     +----------------+
        SJC broadcaster --------- pulled to AMS edge ---------> European viewer
```

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" PROVEN CONFIG-FUTURE

The iroh origin→edge dial is now the mechanism (§7.5, §9), not a future one. The DHT record's `origin=` value is the origin relay's iroh EndpointId, and the edge dials it over iroh (outbound connect + hole-punch) rather than by `host:port`. Because relay↔relay dials by key, a fleet can be an **origin with zero inbound ports** — no port-forwarding, no public IP — behind home or corporate NAT, and edges still reach it. What remains forward-looking is the *fully self-hosted* variant below (your own iroh relay + your own DHT), not the key-dialling itself.

- **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 token). Geo request-routing to the nearest edge (anycast / geoDNS) is deliberately **out of scope** for now, as is metering and settlement between partners — who accounts for edge egress when one partner's edge serves another's viewer is a real interconnect concern (it is why CDN peering has settlement), left here as a future control-plane matter.

Control-plane precedent. `tinymoq.com/cdnadmin` already has fleets registering up to a central `cdnmanager` that holds the customer registry. The planned handoff: `cdnmanager` routes a request to a downstream fleet and passes *(token + origin)* so the edge can pull from the origin. The iroh change is simply that **origin becomes a NodeID instead of a `host:port`**, so an edge can pull from an origin that has no public address (§14).

> **Open question — not decided.** In the shared graph, when a Vivoh viewer watches a Wallflower stream, *who mints the view token?* Content belongs to the origin customer and access is customer-owned, so the **origin (Wallflower) must mint or delegate the view token** even though the viewer arrived via Vivoh's edge. Two paths: **public streams** — the origin mints an open/public view token and anyone on the graph watches (expected common case); **gated streams** — `cdnmanager` brokers a cross-customer token request (viewer's site asks, origin's policy answers). The gated path is heavier and is deferred.

> **Open question — the harder half (confidentiality).** The wrinkle above concerns *access* (B); end-to-end encryption raises a second, harder one. An authorized cross-partner viewer handed a valid view token still holds only *ciphertext* — they also need the origin's decryption **key** (concern C), and key distribution across trust domains is the genuinely hard part of any E2E system. E2E is in fact in *tension* with the open graph: the graph's whole appeal is "any partner's viewer watches any partner's stream," yet E2E means only key-holders decode. Two things make C's version harder than B's:
>
> - A **token** can be brokered and then discarded — short-lived, scoped, revocable. A **key** cannot: possession is permanent, so revoking access means *rekeying the stream*, not tearing up a token.
> - The cross-org handoff for C therefore needs forward-secrecy / key-rotation thinking that B does not.
>
> **Public streams dissolve it** (no encryption, or a published key — anyone decodes). **Gated cross-org streams are the real unsolved problem** — the honest research core of the whole design.

16. Signed records — why open discovery is trustworthy LIVE

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.

17. Status — proven vs. not yet built

Capability	Status	Where
Watch-by-pubkey, browser to browser (mint key → browser signs & PUTs a pkarr record to the public Mainline DHT → a second browser resolves it off the DHT → watch)	**Live**	Wallflower.tv ↔ multicast.video (`?node=` path); no directory server in the path
Cross-fleet edge pull from the resolved origin — real fmp4 media across two independent fleets	**Proven**	2026-07-12 and again browser-to-browser July 2026 (`ams.moqcdn.net` + `ams.gpcmoq.com`)
Self-authenticating discovery — DHT accepts a write under a key only if signed by that key (no registration authority)	**Live**	pkarr/BEP44 on the public Mainline DHT (§16)
Iroh *transport* origin→edge — edge dials the origin relay's EndpointId over iroh (NAT-friendly origin, no public port)	**Proven**	§7.5, §14 — the DHT record's `origin=` carries the EndpointId
Hermit bulk-RX fix — a Hermit relay sustains bulk media on a connection it *dialled out* (prior `smoltcp` RX-buffer depth + cross-core wakeup bugs, now fixed)	**Proven**	3-box WAN Hermit mesh, ~25 Mbps fleet
Two provisioning modes — *direct* (app → box `/assign` , `tmq_` bearer) and *brokered* (app → Worker `/api/publish` / `/api/edge` , `cdn_` token → `/cdn/assign`), interoperating via the DHT	**Live**	tinymoq Worker + fleet boxes; content-blind either way
Relay-blind MoQplay flow — token-gated, end-to-end-encrypted, via the `/assign` broker	**Live**	Wallflower.tv — the encrypted product flow (§7.2, §5)
Cross-org *gated* streams — a Vivoh viewer watching a *gated, encrypted* Wallflower stream (cross-domain view token + key distribution)	**Not built**	§15 — the honest research core; public/open streams already work
Fully self-hosted "private island" — your own iroh relay + your own DHT, peering only with chosen partners	**Config-future**	§14 — the key-dialling itself is done; this is the deployment variant
Geo request-routing (anycast / geoDNS)	**Deferred**	deliberately out of scope for now

Part II in one line

iroh moves up the stack — the browser↔edge media hop is always WebTransport, the origin→edge hop rides iroh (by EndpointId) or native QUIC (by `host:port`), and identity + discovery are a self-signed record on the public Mainline DHT with no server — so **scalable** (pooled Hermit fleets), **decentralized** (server-less DHT findability), and **secure** (customer-held access tokens + relay-blind encryption on the MoQplay flow) each live in their own plane and stop competing.