igc-net: An Open Peer-to-Peer Protocol for Publishing, Discovering, and Exchanging IGC Flight Files

Norwegian University of Science and Technology
Distributed Systems Engineering Lab
Norway *

Draft 0.1 — April 2026

Abstract

Paragliding, hang-gliding, and sailplane communities rely on IGC (International Gliding Commission) flight log files as the canonical record of a soaring flight. Today these files are uploaded to isolated platform silos (XContest, Leonardo, DHV-XC, SeeYou Cloud, and others) with no interoperable mechanism for cross-platform discovery, exchange, or long-term archival. A best-effort public census completed on 9 April 2026 identified at least 16 active public portals and services handling IGC upload, viewing, scoring, or storage. *igc-net* addresses this gap by defining a neutral peer-to-peer interoperability and archival layer beneath existing portals and tools. The protocol gives every IGC file a universal content-addressed identity, enables portals and nodes to publish, discover, and fetch flight logs without central coordination, and allows optional metadata and derived analytics to be attached to the same shared flight identity. This creates a common exchange substrate on which scoring portals, visualisation tools, archives, clubs, researchers, and new analytics services can interoperate without bilateral integrations everywhere. Because each published artifact carries geographic context, local, regional, or national archives can be built automatically, and flights retained by archival nodes remain retrievable as long as at least one archival node continues to serve them. We describe the protocol design, the current reference implementation, and the practical ways in which igc-net can support cross-platform exchange, open experimentation, and long-term preservation of soaring flight data.

1 Introduction

The IGC file format [1], standardised by the International Gliding Commission of the Fédération Aéronautique Internationale (FAI), is the universal data format for recording soaring flights. Every modern flight instrument, from dedicated GPS loggers to smartphone applications, produces IGC files. These files encode timestamped GPS fixes, barometric altitude, pilot identity, glider information, and optional sensor data in a compact, line-oriented text format. IGC files serve as the evidentiary basis for competition scoring, badge claims, safety analysis, and the growing field of soaring meteorology.

Despite this universality at the file-format level, the ecosystem for *exchanging* IGC files is deeply fragmented. A pilot who completes a cross-country flight typically uploads the resulting IGC file to one or more web platforms: XContest, DHV-XC, Leonardo, flightlog.org, SeeYou Cloud, OLC. In addition, some pilots have to also submit their IGC files to a growing number of specialised applications: SkyVarg[1] (AI-powered thermal and XC analysis), PG Pilot[2] (mobile

*Corresponding author: mariusz.nowostawski@ntnu.no
[1] https://skyvarg.ai
[2] https://pgpilot.app

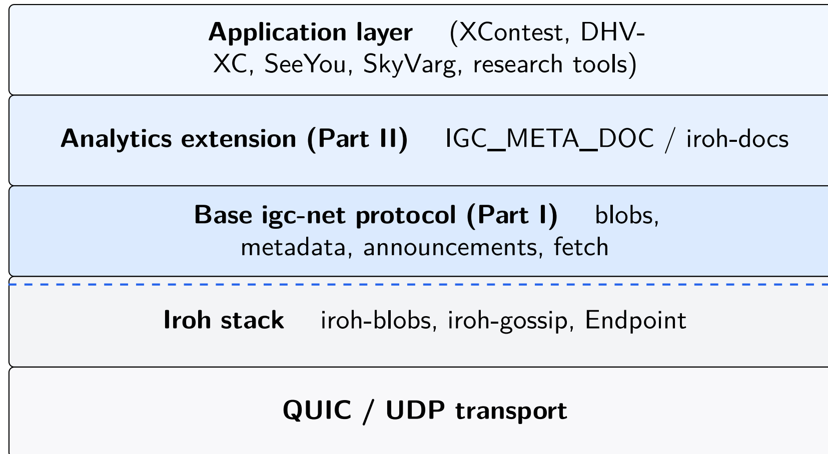

Figure 1: igc-net protocol layer stack. igc-net defines only the application-level semantics (shaded blue); transport and content addressing are provided by the Iroh stack (grey).

logbook and social platform), SkyViz[3] (3D flight visualisation), IGC Viewer[4] (BGA Ladder's web-based track viewer), Flyskyhy, XCTrack, and others. Each of these platforms ingests, parses, and indexes the file independently. These platforms provide valuable services: scoring, visualisation, social features, and statistical analysis. Unfortunately the underlying data, the raw IGC bytes themselves, are locked inside each platform's proprietary storage. There is no way for platform A to discover that platform B holds the same flight, no universal identifier for a flight log, and no mechanism for a third party to request the raw bytes without going through a platform-specific API, complex agreements or (often undesirable) scraping.

igc-net, an open protocol addresses this gap. While the protocol applies to all IGC-producing disciplines (paragliding, hang-gliding, sailplane), this paper uses paragliding examples for concreteness. *igc-net* defines a neutral, content-addressed exchange layer positioned *below* the value-added services that platforms provide. Its purpose is not to replace soaring platforms but to give every IGC file a universal identity and make it discoverable and fetchable across the entire soaring ecosystem.

A portal-specific API can improve access to one portal's own data, but it does not by itself solve ecosystem-wide interoperability. Each platform API carries that platform's identifiers, schemas, access policies, and operational constraints. igc-net instead targets the coordination problem directly by defining a shared protocol layer that multiple portals, archives, and tools can use in common, rather than requiring separate bilateral integrations across the ecosystem.

igc-net achieves this by giving every IGC file a deterministic, content-addressed identity (BLAKE3), disseminating announcements over a gossip overlay without central coordination, and attaching structured metadata, including a geographic bounding box, that enables geographic filtering for local, regional, or national archives. The protocol is specified in implementation-independent normative documents, so that independent implementations in any language can interoperate from the specification alone. The design principles that guided these choices are presented in Section 4.

2 Problem Statement

The scale of the problem is substantial. XContest alone logs approaching 500,000 flights per season [10]; DHV-XC, OLC [11], Leonardo, and other platforms add further volume. The global free-flying community is estimated at over 100,000 active pilots across national federations (based

[3] https://skyviz.io
[4] https://igcviewer.bgaladder.net

on CIVL member-federation membership reports), with additional participants in sailplane disciplines. A typical IGC file is 50–200 KB, placing the annual corpus on the order of 50–100 GB of raw flight data—modest by modern storage standards, but fragmented across at least 16 active public portals and services identified in the April 2026 project census.

The current IGC file ecosystem suffers from several structural problems that igc-net is designed to address.

2.1 Data Silos

When a pilot uploads an IGC file to XContest, the raw bytes and all derived metadata become part of XContest's private data store. If the same pilot also wants the flight visible on DHV-XC, they must upload the same file again to a separate endpoint. If a researcher wants access to the raw bytes, they must negotiate with each platform individually. The result is that the same flight exists as multiple independent copies with no shared identity and no mechanism for cross-reference.

The proliferation of specialised platforms deepens this fragmentation. SkyVarg applies machine learning to thermal and XC route analysis but requires its own flight upload. PG Pilot provides a mobile-first social logbook. SkyViz offers immersive 3D flight visualisation. The BGA Ladder's IGC Viewer serves the UK sailplane community. OLC operates the largest gliding XC competition globally. Each application ingests IGC files independently, re-derives metadata, and stores results in its own database. A pilot who wants their flight analysed by all of these services must upload the same file to each one separately.

2.2 Single Points of Failure

Each platform's upload endpoint is an availability bottleneck. When XContest is down, flights cannot be published to XContest. When a platform shuts down permanently, as has happened with several smaller scoring services, all flights stored exclusively on that platform are lost. There is no community-operated, platform-neutral archive.

2.3 No Canonical Content Identity

The same raw IGC bytes, uploaded to three different platforms, receive three different platform-specific identifiers (database row IDs, UUIDs, or URL slugs). There is no universal, platform-independent way to determine that these three records describe the same physical flight. Deduplication, cross-referencing, and citation are all impossible at the protocol level.

2.4 No Open Metadata Exchange

Every platform that ingests an IGC file independently re-derives the same basic metadata: flight date, pilot name, glider type, bounding box, altitude envelope, fix count. This duplicated effort is wasteful and produces inconsistent results when parsers differ in edge-case handling. There is no standard, shareable metadata blob that platforms could exchange.

2.5 No Open Analytics Layer

Derived products, such as e.g. thermal detection, wind estimation, flight-phase segmentation, site matching, XC scoring, are among the most valuable outputs of the soaring data ecosystem. Today these analytics are locked inside each platform's proprietary stack. A thermal map computed by one platform cannot be consumed by another. Researchers who want to study thermal distribution must either build their own analysis pipeline from raw IGC files or negotiate data access with each platform separately.

The emergence of AI-driven flight analysis, automated thermal detection, route optimisation, safety-risk scoring, and natural-language flight debriefing, accelerates this problem. Each AI tool trains on and produces results for its own data silo. Without an open exchange layer, the soaring community cannot benefit from composing multiple AI analyses on the same flight, nor can researchers build on each other's derived products.

2.6 Archival Gap

The soaring community has no long-term, platform-neutral archive of flight data. Individual pilots may keep personal backups, but there is no systematic mechanism for preserving the collective corpus. Historical flights from defunct platforms are irrecoverably lost. The absence of an archival layer is particularly damaging for longitudinal research in soaring meteorology, where decades of flight data could inform climate and atmospheric models.

2.7 No Local or National Flight Archives

Many national federations (DHV in Germany, FFVL in France, BHPA in the UK), regional associations, and individual flying clubs have a legitimate interest in maintaining an archive of flights conducted in their jurisdiction. Today, building such an archive requires either bilateral agreements with every international platform or asking every pilot to manually upload to an additional national endpoint. Neither approach scales. There is no mechanism for a national body to say "give me every flight whose bounding box overlaps my country" and have the raw data arrive automatically from the global corpus.

3 Historical Background

igc-net began in 2023 at NTNU's Distributed Systems Engineering (DSE) Lab as an effort to build an open exchange layer for soaring flight data.

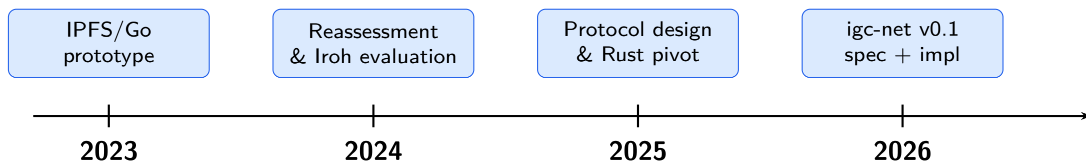

Figure 2: igc-net project timeline (2023–2026).

3.1 Early Exploration: IPFS and Go (2023)

The first prototype used Go and IPFS [4]. It validated the basic idea of content-addressed flight exchange, but it exposed two mismatches for this domain: IPFS's DHT performed poorly in a small, seasonal network, and the operational footprint was too high for easy adoption by soaring platforms.

3.2 Reassessment (2024)

In 2024, the project evaluated alternative P2P stacks against five criteria: deterministic content addressing, lightweight gossip without global DHT dependency, low operational overhead, a memory-safe embeddable implementation, and active upstream maintenance. The Iroh project (n0.computer) emerged as the strongest candidate.

3.3 Protocol Design and Rust Pivot (2025–2026)

With Iroh selected, the project made two architectural decisions:

1. **Protocol-first design.** The protocol specification was written as a set of implementation-independent normative documents using RFC 2119 keywords. The specification defines wire formats, content identifiers, gossip topics, metadata schemas, and fetch semantics without reference to any specific programming language.

2. **Rust reference implementation.** A Rust implementation was chosen as the reference implementation due to Iroh's native Rust API, the Rust ecosystem's strengths in systems programming, and the language's suitability for embedding as a library crate in larger applications.

The specification was separated into its own repository, and the Rust implementation was developed in parallel against it. By early 2026, the project had a working specification and a complete Rust reference implementation for Part I. Part II remains specified but not yet implemented.

4 Design Principles

The igc-net protocol is guided by the following design principles.

Content addressing as the universal identifier. Every IGC file and every metadata blob is identified by the BLAKE3 hash of its exact bytes. This gives each artifact a deterministic, platform-independent identity. Two nodes that independently hash the same raw IGC bytes will always arrive at the same `igc_hash`, regardless of when, where, or by whom the hashing was performed.

Protocol-first. The normative specification is written in plain language with RFC 2119 [7] keywords and contains no language-specific APIs, types, or code. Any developer can implement a compliant igc-net node from the specification alone, in any language, without consulting the Rust reference implementation. The specification is the source of truth; the Rust code is informative.

Minimal mandatory surface. The base metadata schema requires exactly three fields: `schema`, `schema_version`, and `igc_hash`. All other metadata is optional: pilot name, glider type, bounding box, altitude envelope. This ensures that even the most minimal implementation can participate in the network.

No central registration. Any node that can reach the gossip overlay can publish or index flights. There is no authority that approves nodes, assigns identifiers, or curates content. The network is open by default.

Separation of base protocol and analytics. The base protocol handles raw IGC bytes and basic metadata. Derived analytics such as thermals, wind, scoring, are handled by an optional extension layer that can evolve independently. A node that implements only the base protocol remains fully compliant.

Public network. Publishing to igc-net is a public act. Any node that joins the network can observe announcements and attempt to fetch announced content. The protocol does not provide confidentiality or access control at the transport layer. Privacy, if required, is an application-layer concern.

Table 1: Published flight blob pair.

Blob	Content	Hash field
IGC blob	Raw `.igc` file bytes, unmodified	`igc_hash`
Metadata blob	UTF-8 JSON conforming to the base schema	`meta_hash`

5 Protocol Architecture

This section describes the core protocol architecture. The normative specification is split across three documents: `specs_igc.md` (wire format, gossip, fetch flow), `specs_meta.md` (metadata schema, analytics extension), and `requirements.md` (functional and non-functional requirements). This section synthesises the key elements.

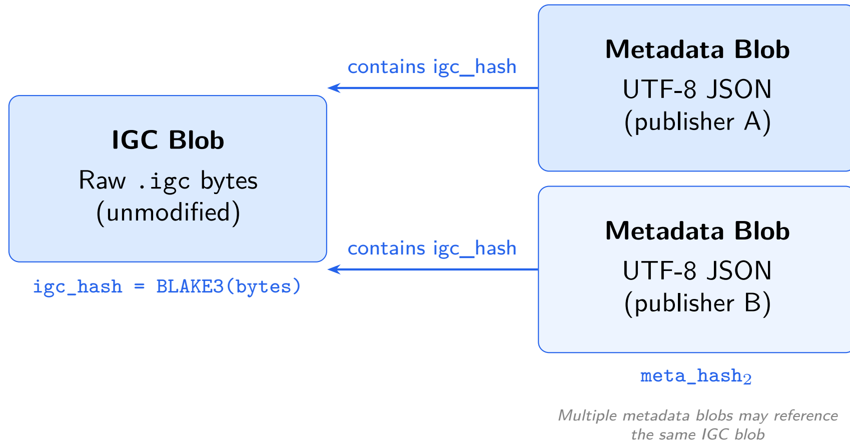

Figure 3: Two-blob model for a published flight. Each metadata blob contains the `igc_hash` as a required field, binding it cryptographically to the IGC file. Multiple publishers may produce different metadata blobs for the same flight.

5.1 Blob Model

Every published flight consists of exactly two content-addressed blobs:

Both blobs are addressed by their BLAKE3 hash, encoded as 64-character lowercase hexadecimal strings. The pair (`igc_hash`, `meta_hash`) identifies a specific published version of a flight on the network.

The IGC blob contains the exact bytes of the original `.igc` file, unmodified. No normalisation, re-encoding, or transformation is applied before hashing. This ensures that two publishers who received the same file from the same instrument will produce the same `igc_hash`.

The metadata blob is a UTF-8 JSON object that conforms to the base metadata schema (Section 5.4). It is serialised, hashed, and stored as a separate content-addressed blob. Because the metadata blob includes the `igc_hash` as a required field, it is cryptographically bound to the IGC file it describes.

All content addressing uses a cryptographic hash function that produces 32-byte (256-bit) digests [3]. Each igc-net node holds an Ed25519 [5, 8] key pair whose 32-byte public key serves as the node's stable network identity. The secret key must persist across restarts; implementations must not generate a new identity on every startup. In wire formats, public keys are encoded as 64-character lowercase hexadecimal strings.

Immutability. Once a blob is published, its content cannot be changed without changing its hash. Corrected metadata must be published as a new metadata blob with a new `meta_hash`. The

Table 2: Announcement message fields.

Field	Type	Description
`igc_hash`	string	BLAKE3 hash of the IGC blob
`meta_hash`	string	BLAKE3 hash of the metadata blob
`node_id`	string	Serving node's Ed25519 public key
`igc_ticket`	string	Iroh `BlobTicket` for the IGC blob
`meta_ticket`	string	Iroh `BlobTicket` for the metadata blob

protocol explicitly permits multiple metadata blobs for the same `igc_hash`, allowing independent publishers to provide different levels of metadata completeness.

5.2 Gossip and Announcements

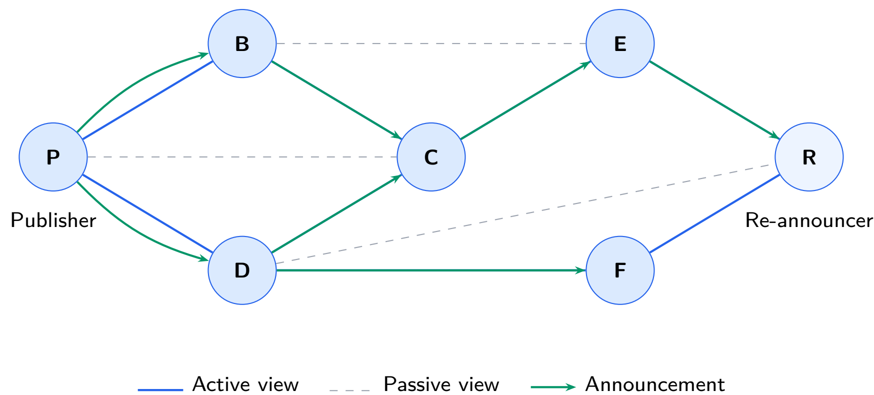

Figure 4: HyParView gossip overlay and announcement propagation. Node P publishes a flight; the announcement propagates through active-view edges. Node R re-announces with its own `node_id` but the same hashes. Dashed lines show passive-view entries used for failover.

igc-net uses a gossip overlay for announcement dissemination. All compliant nodes join a single well-known *announce topic* whose identifier is derived deterministically:

```
topic_id = BLAKE3("igc-net/announce/v1")
```

The canonical hexadecimal value is:

`2f06567e5d7148b56349a753f8b407fbc35b2f0d90ff366ff2673d060245dda9`

Implementations embed this value as a compile-time constant. The derivation string encodes the protocol major version; breaking wire-format changes produce a new derivation string and therefore a new topic.

Announcement message. An announcement is a compact UTF-8 JSON object that asserts: "I hold both blobs for this flight and can serve them."

The message must not exceed 1024 bytes. Receivers discard oversized or internally inconsistent announcements without protocol response. Unknown fields are ignored, enabling forward-compatible extensions. Three consistency constraints bind the announcement fields together:

1. The hash embedded inside `igc_ticket` must equal `igc_hash`.
2. The hash embedded inside `meta_ticket` must equal `meta_hash`.
3. The node identity embedded in both tickets must equal `node_id`.

Re-announcement. Any node that has fetched and hash-verified both blobs may re-announce the flight using its own tickets and `node_id`. The `igc_hash` and `meta_hash` remain unchanged; only the serving node's identity and tickets change. This mechanism provides natural replication: as more nodes fetch and re-announce a flight, the number of serving peers grows, improving availability and fetch resilience.

The deduplication key for announcements is the pair (`meta_hash`, `node_id`). The same `meta_hash` from a different `node_id` is treated as a distinct serving peer, not a duplicate.

Re-announcement amplification. In a network with N archival nodes, a single published flight may generate up to N re-announcements. Each receiving node performs a constant-time dedup lookup per re-announcement, bounding the per-node cost to $O(N)$ hash-set checks per flight. At current anticipated network sizes (tens to low hundreds of archival nodes) this is acceptable.

BlobTicket coupling. The v1 announcement format uses Iroh BlobTicket serialisations as the transfer mechanism. A non-Iroh implementation must either parse the Iroh ticket format or implement an alternative fetch path. This is a deliberate trade-off: Iroh tickets embed the serving peer's address and the blob's hash in a single opaque string, simplifying the announcement format. A future protocol version may define a transport-agnostic fetch URI scheme to reduce the Iroh coupling, however this will introduce transport implementation complexity on the nodes.

5.3 Discovery and Fetch Flow

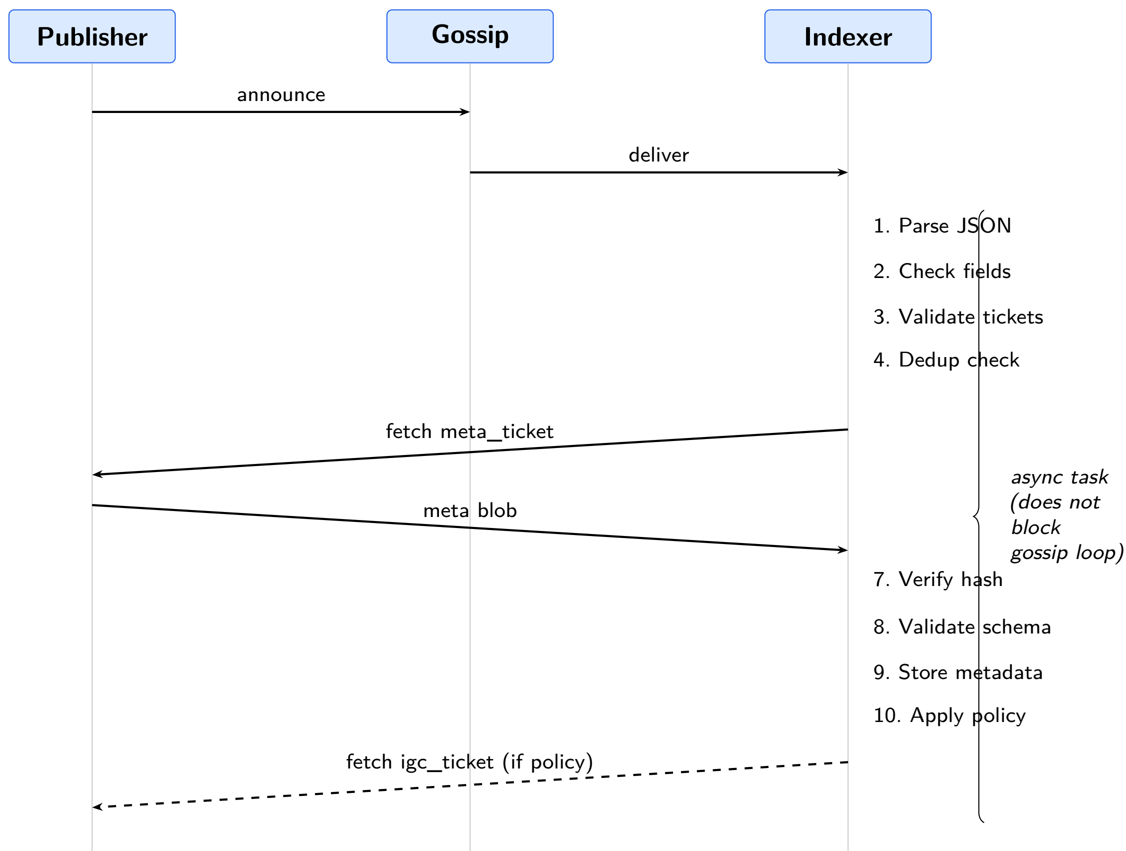

Figure 5: Indexer fetch pipeline (11-step sequence). Steps 5–11 run in a spawned async task so the gossip receive loop is never blocked. The dashed arrow indicates a conditional fetch governed by the local policy.

On receiving an announcement, a node processes it through an ordered pipeline:

1. **Parse** the JSON payload.
2. **Discard** oversized or malformed payloads, or those missing required fields.
3. **Validate** ticket/hash/node consistency.
4. **Deduplicate** using (`meta_hash`, `node_id`).
5. If `meta_hash` is already known but `node_id` is new, **record the new fetch source** and stop.
6. **Fetch** the metadata blob via `meta_ticket`.
7. **Verify** BLAKE3(metadata_bytes) = `meta_hash`.
8. **Validate** the metadata blob: UTF-8 JSON, correct schema identifier, supported schema version, matching `igc_hash`, canonical timestamps, valid `flight_date` if present, finite in-range coordinates, and ordered bounding-box bounds.
9. **Store** the validated metadata locally.
10. **Apply** the local raw-blob fetch policy.
11. If fetching the raw IGC: fetch via `igc_ticket`, verify BLAKE3(igc_bytes) = `igc_hash`, store only if verified.

A critical implementation requirement is that metadata and raw-blob fetches must not block the gossip receive loop. The reference implementation achieves this by spawning each announcement's fetch pipeline as an independent asynchronous task.

Fetch policies. The protocol defines three recommended fetch policies:

MetadataOnly: Fetch and store only the metadata blob. The raw IGC blob is fetched on explicit request.

Eager: Fetch and store both the metadata blob and the raw IGC blob for every announced flight.

GeoFiltered: Always fetch the metadata blob (as with all policies), then fetch the raw IGC blob only when the metadata's bounding box overlaps a configured geographic region. The geographic filter is applied after metadata retrieval, not at announcement time.

5.4 Base Metadata Schema

The base metadata schema is identified by the literal pair:

```
schema = "igc-net/metadata"
schema_version = 1
```

It defines three required fields and approximately twenty optional fields:

Serialisation and validation rules. Optional fields that are unavailable must be omitted from the JSON, not set to `null`. Timestamps use the canonical format `YYYY-MM-DDTHH:MM:SSZ`. Hashes and node IDs use lowercase hexadecimal. If present, `flight_date` must be a real `YYYY-MM-DD` calendar date; coordinates must be finite and within WGS84 latitude/longitude bounds; and `bbox` must satisfy `min_lat` ≤ `max_lat` and `min_lon` ≤ `max_lon`. Publishers may add publication-context fields such as `original_filename`, `publisher_node_id`, and `published_at`, but must not fabricate flight-derived values not derivable from the IGC source file.

Table 3: Required metadata fields.

Field	Type	Constraints
schema	string	Must equal "igc-net/metadata"
schema_version	integer	Must equal 1
igc_hash	string	64-char lowercase BLAKE3 hex

Table 4: Selected optional metadata fields (full reference in Appendix B).

Field	Type	Source
flight_date	string	HFDTE header record
started_at	string	First valid B-record timestamp
ended_at	string	Last valid B-record timestamp
duration_s	integer	Derived from timestamps
pilot_name	string	HFPLT header record
glider_type	string	HFGTY header record
bbox	object	Min/max lat/lon over B-records
max_alt_m, min_alt_m	integer	Altitude envelope
publisher_node_id	string	Publishing node's public key
published_at	string	Publication timestamp

Multiple metadata blobs. The protocol intentionally leaves metadata selection to the consumer. Different applications have different fidelity requirements: a scoring platform may prefer the metadata blob with the most populated fields, while an archival node may prefer the earliest `published_at` for provenance. The recommended default heuristic, "prefer more populated fields, then later `published_at`," serves as a sensible starting point, but consumers are free to implement application-specific selection logic.

Schema versioning. `schema_version` is a positive integer. Unsupported versions are ignored without protocol response. Additive optional fields may be introduced without incrementing the version if they do not break existing parsers. Breaking semantic changes require a version increment.

5.5 Analytics Extension (Part II)

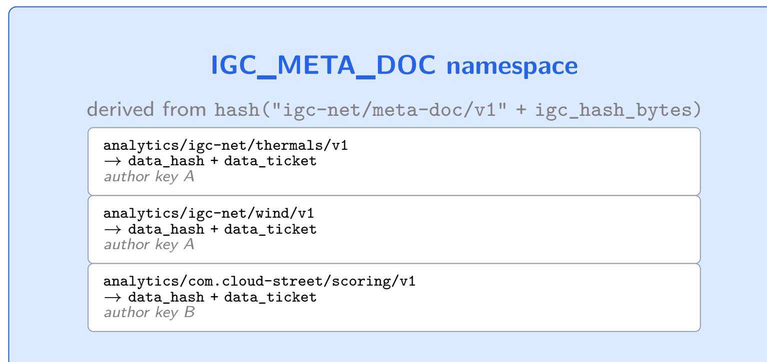

Figure 6: Analytics extension architecture for a single flight. Each entry in the `IGC_META_DOC` namespace points to a content-addressed analytics blob. Multiple authors can write to the same document.

The optional analytics extension, called `IGC_META_DOC`, defines how derived analytics for a published flight can be published, discovered, and exchanged. A node implementing only the base protocol remains fully compliant; the analytics extension is strictly optional.

Table 5: Standard analytics type identifiers.

Type	Identifier
Flight classification	`igc-net/classification/v1`
Site matching	`igc-net/sites/v1`
Thermal detection	`igc-net/thermals/v1`
Wind estimation	`igc-net/wind/v1`
Flight-phase segmentation	`igc-net/segments/v1`

Deterministic identifiers. The analytics layer uses its own gossip topic, derived from `BLAKE3("igc-net/analytics/v1")`. Each flight has a deterministic `IGC_META_DOC` namespace derived from its `igc_hash`:

```
namespace_secret = hash("igc-net/meta-doc/v1" + igc_hash_bytes)
```

Because this derivation is deterministic and based on public data, writes to the document are protocol-open. Attribution and trust are handled at the application layer.

Standard analytics types. The protocol defines a namespace for standard analytics types using the `igc-net/` prefix:
Custom analytics types use reverse-DNS namespacing (e.g., `com.cloud-street/scoring/v1`, `org.xcontest/pg-daily/v1`).

Document structure. Each flight's `IGC_META_DOC` is an `iroh-docs` replicated document. Entries are keyed by `analytics/{analytics_type}` and point to content-addressed analytics blobs rather than embedding payloads inline. Every write is attributable to the writer's author key, enabling consumers to build trust policies over analytics providers.

6 Iroh Foundations

igc-net is built on the Iroh networking stack developed by n0.computer [2]. This section describes how the protocol maps onto Iroh's primitives and why Iroh was chosen over alternatives.

6.1 The Iroh Stack

Iroh provides four composable primitives: a QUIC-based endpoint [9] for peer identity and connectivity, a BLAKE3-addressed blob store (iroh-blobs), a HyParView [6] gossip overlay (iroh-gossip), and an optional replicated document store (iroh-docs). Table 6 maps igc-net concepts to these primitives.

6.2 Mapping igc-net to Iroh

The mapping is deliberate and direct. igc-net does not introduce its own networking, transport, or content-addressing layers. It defines *only* the application-level semantics: which blobs exist, what they contain, how they are announced, and how they are validated.

6.3 Why Iroh over IPFS

The 2023 IPFS prototype (Section 3) informed the decision to adopt Iroh. In 2024, the project evaluated alternative P2P stacks against five criteria: deterministic content-addressed blob storage, lightweight gossip without global DHT dependency, low operational overhead (single binary, minimal configuration), a memory-safe language suitable for embedding, and active upstream maintenance. Iroh satisfied all five. The specific advantages over IPFS are:

Table 6: Mapping igc-net concepts to Iroh primitives.

igc-net concept	Iroh primitive
Node identity (Ed25519 key pair)	`Endpoint` identity
IGC blob storage and transfer	`iroh-blobs` + `BlobTicket`
Metadata blob storage and transfer	`iroh-blobs` + `BlobTicket`
Announcement dissemination	`iroh-gossip` topic subscription
Topic ID derivation	BLAKE3 hash of a canonical string
Per-flight analytics document	`iroh-docs` (planned)
Content verification	BLAKE3 hash comparison

Simpler content addressing. Iroh uses BLAKE3 [3] exclusively. There is no multihash prefix, no multicodec negotiation, and no CID versioning. BLAKE3 provides 256-bit security with performance exceeding SHA-256 and BLAKE2b on modern hardware. This single-algorithm design eliminates an entire class of interoperability edge cases and makes compile-time constants feasible.

No global DHT dependency. IPFS's content routing relies on a global Kademlia DHT that requires a critical mass of peers to function reliably. Iroh's gossip model (HyParView) provides reliable message delivery within a topic without depending on a global routing table. This is a better fit for a niche community with hundreds to thousands of participants.

Lower operational overhead. An Iroh node is a single Rust binary with minimal configuration. There is no separate daemon process, no garbage-collection tuning, and no DHT bootstrap node maintenance.

Rust-native API. Iroh's Rust API enables igc-net to be consumed as a library crate (`igc-net`) that soaring platforms can embed directly into their backend services. The Go IPFS libraries required CGo interop or HTTP API calls for non-Go consumers.

QUIC transport. Iroh uses QUIC [9] natively, providing encrypted connections, multiplexed streams, and hole-punching without the TLS and TCP overhead of IPFS's libp2p transport stack.

7 Reference Implementation

A complete Rust reference implementation for Part I accompanies the protocol specification. It is developed as a separate repository (`igc-net-rs`) that tracks the specification documents. The broader project also ships adjacent Python bindings in `igc-net-py`, but those bindings are not part of the Cargo workspace described here.

7.1 Workspace Structure

The implementation is a Cargo workspace with two members:

`igc-net` — the library crate, implementing the core protocol logic. This is the package that soaring platforms embed as a dependency.

`igc-net-cli` — a standalone command-line binary (`igc-net`) that provides a reference archival node and offline inspection tools.

Table 7: Library module architecture.

Module	Responsibility
id	Typed identifier newtypes for BLAKE3 hashes and node IDs
topic	Well-known gossip topic ID constants and derivation verification
metadata	FlightMetadata struct, IGC parsing, validation, serialisation
store	Content-addressed flat-file blob store, append-only index, in-memory caches, key management
node	IgcIrohNode runtime: endpoint, blobs, gossip, router lifecycle
publish	Publish pipeline: hash, build metadata, store, ticket, announce
indexer	Gossip listener, announcement validation, fetch pipeline, rate limiting
util	Shared validation and formatting helpers

7.2 Module Architecture

The library crate is organised into eight source modules:

Each module has a clearly bounded responsibility. The publish and indexer modules implement the publisher and indexer pipelines described in Section 5, including rate limiting, deduplication, and async task spawning. The store module provides $O(1)$ deduplication lookups via in-memory caches populated at startup. The public API also exposes typed identifier newtypes and a thiserror-based error hierarchy rather than erasing failures into generic error values.

7.3 CLI Reference Node

The igc-net binary provides five commands:

Table 8: Reference CLI commands.

Command	Description
igc-net announce <file.igc>	Publish an IGC file to the network; --linger <secs> keeps the node alive temporarily after publish
igc-net runindex	Run an indexer node; supports --policy <p>, --bootstrap <ids>, and --peer-addr <addr>
igc-net fetch <igc_hash>	Export a locally stored IGC blob; --out <file> overrides the default output path
igc-net inspect <file>	Offline inspection of an IGC or metadata file
igc-net list	List all flights in the local index

The data directory defaults to $HOME/.igc-net and can be overridden via --data-dir or the IGC_NET_DATA_DIR environment variable.

7.4 Test Strategy

The reference implementation employs a two-tier testing strategy:

- **Unit tests** (50 tests) cover every module: topic derivation and spec-constant verification, metadata construction and validation (including edge cases such as midnight-crossing

timestamps, invalid dates, out-of-range coordinates, invalid bounding boxes, and uppercase hash rejection), store round-trips and deduplication, announcement serialisation and size checks, and announcement validation for malformed or oversized inputs.

- **Integration tests** (5 tests) start real Iroh nodes on loopback and exercise the full publish-index-fetch pipeline across multiple nodes. Tests cover MetadataOnly indexing, Eager indexing with content integrity verification, multi-publisher re-announcement with fetch-source retention, GeoFiltered fetch policy, and deduplication of duplicate announcements.

7.5 Preliminary Measurements

The following analytical estimates characterise the protocol's resource profile. Empirical multi-node measurements will be reported in a future evaluation; the values below are derived from the reference implementation and HyParView's known properties.

Table 9: Protocol resource estimates.

Metric	Estimate
Metadata blob size	400–800 bytes
Announcement JSON size	300–500 bytes
Raw IGC blob size	50–200 KB
Storage per flight (metadata-only)	∼1 KB
Storage per flight (eager archival)	50–200 KB
Annual corpus (eager, ∼500k flights)	∼50–100 GB

Gossip overhead estimate. HyParView maintains an active view of ∼5 peers and a passive view of ∼30. Each announcement is forwarded to the active view, so per-node outbound cost is $O(\text{active_view_size})$ per announcement. At 100 peers, 1,000 flights/day yields ∼5,000 gossip messages/day per node. At 10,000 peers the message count per node remains the same (HyParView's active view is constant), though the total network traffic scales linearly with peer count.

8 Deployment Scenarios

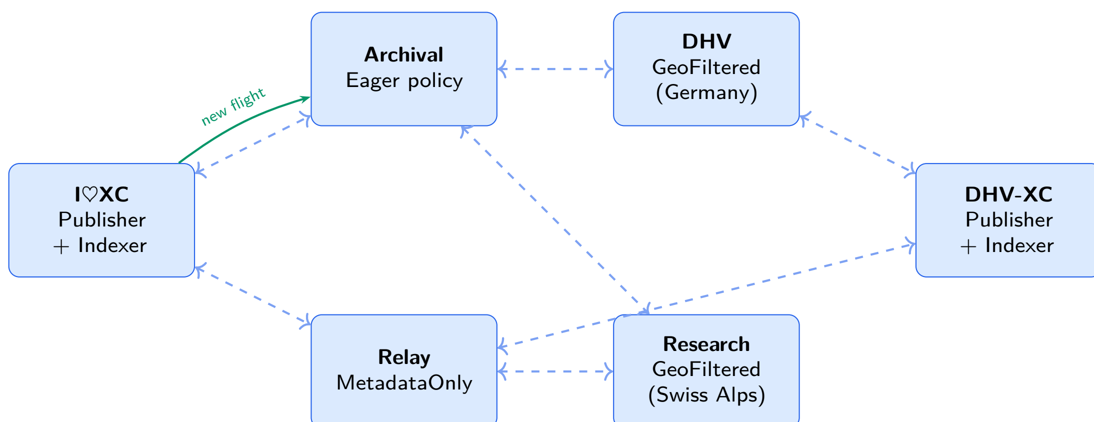

Figure 7: Deployment topology with five roles. Dashed lines show gossip connections; the green arrow shows a blob transfer for a newly published flight. The research node and federation node each apply geographic filters on metadata before fetching raw blobs.

igc-net is designed to support a range of deployment scenarios, from single-operator archival nodes to full platform integration.

The protocol is also intended for incremental adoption. If igc-net only became useful after every major portal adopted it, it would have little practical chance of deployment. The design therefore aims to create value at the edges first: for archives, university groups, researchers, small analytics tools, local clubs, and pilot communities that can already benefit from shared identity, fetch, and archival before the largest incumbents participate. Established platforms may have weaker short-term incentives to support an open exchange layer, since broader interoperability reduces data lock-in and lowers barriers for third parties to build competing analytics and services on top of the same corpus.

8.1 Solo Archival Node

A single operator runs an igc-net node with the Eager fetch policy to mirror the full corpus of announced flights:

```
igc-net runindex --policy eager --bootstrap <peer_id>
```

The node indexes all announcements, fetches and verifies both metadata and raw IGC blobs, and stores them in the local flat-file store. Multiple independent archival nodes operated by different parties provide resilience against data loss.

For pilots, this model behaves like a peer-to-peer archival layer. A pilot can publish once into igc-net and rely on archival nodes to retain the flight beyond any single platform. Availability persists as long as at least one archival node continues to retain and serve the blobs.

8.2 Platform Integration

A soaring platform embeds the `igc-net` library crate directly into its backend. On upload, the platform publishes the IGC file to the igc-net network alongside its normal ingestion pipeline. Concurrently, the platform runs an indexer to discover flights published by other platforms or by pilots directly.

This model allows platforms to:

- **Discover** flights uploaded to other platforms without bilateral API agreements
- **Deduplicate** flights that arrive from multiple sources, using `igc_hash` as the canonical identity
- **Attribute** flights to their original publisher via `publisher_node_id`
- **Offer** richer metadata by merging fields from multiple metadata blobs

8.3 Community Relay

A community-operated node acts as a bootstrap peer and announcement relay. It runs with the MetadataOnly policy to minimise storage requirements while ensuring that all announcements propagate through the gossip overlay.

8.4 Geographic Archival: Federations and Research

The GeoFiltered fetch policy enables any node to build a geographically scoped archive. The node fetches metadata for every announced flight, then downloads the raw IGC blob only when the metadata's bounding box overlaps a configured region.

Federation use case. A national paragliding federation (e.g., DHV, FFVL, BHPA) or a regional flying club runs a GeoFiltered indexer configured with the geographic bounds of its jurisdiction:

```
igc-net runindex --policy geo:47.3,55.1,5.9,15.0 --bootstrap <peer_id>
```

This example covers Germany. The federation gains a complete, continuously updated archive of flights in its territory, regardless of which platform the pilot originally uploaded to, without any data-sharing agreements or site scraping. This model is particularly valuable for:

- **Safety analysis:** studying incident patterns and airspace usage across all flights in the jurisdiction.
- **Site management:** tracking launch-site usage and monitoring compliance with airspace restrictions.
- **National statistics:** reporting total flights, active pilots, and geographic distribution independently of any single platform.

Research use case. A researcher uses the same mechanism to build a curated local dataset:

```
igc-net runindex --policy geo:46.0,48.0,6.0,10.0 --bootstrap <peer_id>
```

This node collects raw IGC blobs only for flights overlapping the Swiss Alps, producing a ready-made corpus for atmospheric research, thermal analysis, or regional flight statistics, built automatically from the network's live traffic.

8.5 University Archival Stewardship

NTNU DSE Lab is part of NTNU, a public university and non-profit research environment. Ongoing work at the lab includes operating archival capacity on university infrastructure to provide long-lived retention for published igc-net content and practical experience with community archival operation.

9 Security and Trust Model

9.1 Threat Model

Table 10: Threat model summary.

Threat	Mitigation	Residual risk
Content tampering	BLAKE3 verification on every fetch	None
Metadata poisoning	No protocol-level mitigation; consumer-side trust policy	High
Sybil flooding	Per-source rate limiting at indexer	Medium
Gossip amplification	Dedup by (`meta_hash`, `node_id`); $O(1)$ lookup	Low–Medium
Eclipse attack	HyParView's passive view provides failover	Low
Denial of service	MB/day rate limit per publisher	Medium

9.2 Content Integrity

Every blob in igc-net is content-addressed by a hash. Receivers verify the hash of every fetched blob against the announced hash. Any blob that fails verification is discarded. This provides strong guarantees against accidental corruption and deliberate tampering.

The IGC file format includes an optional G-record: a cryptographic signature generated by the flight recorder that attests to the integrity of the recorded data. igc-net's BLAKE3 content hash and the IGC G-record are complementary. The content hash ensures that the bytes received over the network are identical to the bytes that were published. The G-record (when present and verifiable) additionally attests that the bytes were produced by a specific recording device and have not been modified since recording. A future protocol extension could expose G-record verification status as a trust signal in the analytics layer.

9.3 Node Identity and Attribution

Each node holds a persistent Ed25519 key pair. The public key appears in announcements as `node_id` and optionally in metadata as `publisher_node_id`. This provides *attribution*. It is possible to determine which node published or re-announced a flight, but not *authentication* in the traditional sense. The protocol does not include a trust registry, certificate authority, or reputation system.

9.4 Trust as Application-Layer Policy

The protocol deliberately does not prescribe trust semantics. Whether a consumer trusts a particular publisher's metadata, analytics, or raw data is an application-layer decision. A platform might:

- Trust only announcements from known publisher node IDs
- Accept metadata from any publisher but flag unverified sources
- Trust analytics only from authors whose keys are in a curated list
- Ignore analytics types from unknown providers

9.5 Flood Protection

The indexer supports configurable per-source rate limiting: **blobs per hour** (maximum announcements per publisher per rolling hour), **megabytes per day** (maximum total bytes per publisher per rolling 24-hour window), and a **trusted node IDs** allowlist exempt from rate limits.

9.6 Privacy Considerations

igc-net is a public network. All announcements are visible to all participants. Publishing a flight reveals the flight's metadata (including pilot name, glider type, and geographic coordinates) to any node on the network. The protocol does not currently support encrypted blobs, selective disclosure, or access control.

10 Limitations and Future Work

10.1 Current Limitations

No storage incentives. The protocol relies on altruistic participation for data persistence. There is no built-in incentive for nodes to store and serve data beyond their own interest.

No privacy layer. The protocol provides no confidentiality, limiting applicability for flights in restricted airspace or by pilots who prefer anonymity.

Analytics extension not yet implemented. Part II of the specification (IGC_META_DOC) is fully specified but not yet implemented in the Rust reference implementation.

Antimeridian handling. The bounding box specification requires `max_lon` $\geq$ `min_lon`, which does not accommodate flights that cross the 180th meridian.

10.2 Open Protocol Questions

- **Multiple-metadata merge rule:** the current default heuristic may need a stricter or more nuanced selection policy.
- **Schema governance:** who approves new standard `igc-net/*` analytics types after the initial release?
- **Machine-readable schema:** should the protocol publish a normative JSON Schema alongside the prose specification?

10.3 Future Directions

Incremental archive sync. A mechanism for a new archival node to efficiently synchronise with existing nodes, rather than waiting for re-announcements of historical flights. This is the top-priority at the moment.

Mobile integration. Direct publication from flight instrument applications (XCTrack, XC Guide, SeeYou Navigator) to the igc-net network, enabling real-time flight sharing without intermediary platforms.

Cross-network bridging. Gateways that bridge igc-net announcements to conventional APIs (REST, GraphQL) for platforms that cannot embed a full igc-net node.

Pilot identity layer. A lightweight mechanism for pilots to claim ownership of flights via signed attestations, without requiring a central identity provider.

10.4 AI and the Innovation Ecosystem

The soaring community is beginning to benefit from AI-driven flight analysis: machine-learning thermal detectors, route optimisers, safety-risk scorers, and natural-language flight debriefing tools. igc-net is positioned to accelerate this trend in several ways.

AI as a consumer of igc-net. ML-based and LLM-based flight analysis tools can consume the igc-net network to access a diverse, cross-platform corpus rather than the subset available from a single platform's upload pipeline. A thermal-prediction model trained on geographically filtered igc-net data has access to flights from every publisher on the network, not only those uploaded to the model developer's own platform.

AI as a producer on igc-net. AI-generated analytics, e.g. automated thermal maps, wind-field estimates, flight classifications, can be published back to the network via the analytics extension (Section 5.5), making them available to every participant.

Composability. igc-net's content-addressed analytics model enables different AI providers to publish complementary analyses for the same flight. A thermal detector from one provider, a wind model from another, and a safety-risk scorer from a third can all be composed by any consumer or a pilot's mobile app without requiring integration between the providers, or complex orchestrations. This makes derived services modular: specialised tools can publish reusable outputs that other tools consume and combine into higher-level services.

Lowered barrier to experimentation and reuse. New applications, analytics services, and research prototypes no longer need to build their own data-ingestion pipelines or negotiate API access with each platform. They can bootstrap from the igc-net network, reducing time-to-market and enabling solo developers, student projects, and researchers to work against a broader shared corpus.

Ecosystem agility. An open data layer makes the soaring ecosystem more robust: no single point of failure for derived analytics, more dynamic: new safety, training, visualisation, and learning tools can appear and consume data immediately, and more reusable: new services can build on prior outputs instead of recomputing everything from scratch. igc-net is designed as a neutral substrate on which the next generation of soaring tools can be built.

11 Related Work

11.1 Peer-to-Peer Systems

IPFS and libp2p. The InterPlanetary File System [4] provides content-addressed storage with a global DHT for content routing. As discussed in Section 3, IPFS's DHT-based routing was a poor fit for igc-net's niche community. libp2p's modular transport layer introduces complexity that is unnecessary when a single transport (QUIC) suffices.

BitTorrent. BitTorrent provides efficient bulk data transfer via swarming, but its content model is oriented around named torrents rather than individual content-addressed blobs. BitTorrent lacks a built-in gossip layer for real-time announcement of new content.

Hypercore/dat. The Hypercore protocol provides append-only logs with content verification. Its replication model is well-suited to ordered streams but less natural for igc-net's unordered collection of independent flights.

Iroh. igc-net builds directly on Iroh [2] (Section 6). Iroh's combination of BLAKE3 content addressing, QUIC transport, HyParView gossip, and replicated documents provides all the primitives igc-net needs without requiring igc-net to implement its own networking layer.

11.2 Domain-Specific Systems

The soaring data ecosystem comprises a growing number of platforms and applications, none of which supports a standardised cross-platform exchange mechanism. The project repository maintains a dated best-effort census (`PORTALS.CSV`) that currently lists at least 16 active public portals and services. Table 11 is illustrative rather than exhaustive.

XContest [10] is the largest paragliding competition and flight-sharing platform. Provides scoring, flight visualisation, and social features. Flights are uploaded via a web interface or API and stored in a centralised database.

DHV-XC is the German Hang Gliding and Paragliding Association's cross-country platform. Similar to XContest in architecture: centralised upload, proprietary storage, no interoperable exchange.

OLC (Online Contest) [11] is the largest gliding XC competition platform globally, operated by Segelflugszene GmbH.

Table 11: Selected examples from the soaring platform landscape (2026). No platform supports standardised cross-platform exchange.

Platform	Type	IGC input	Exchange
XContest [10]	Scoring / social	Upload	None
DHV-XC	Scoring / national	Upload	None
OLC [11]	Scoring (gliding)	Upload	None
Leonardo	Scoring (legacy)	Upload	None
SeeYou Cloud	Analysis / viz	Upload	None
SkyVarg	AI analytics	Upload	None
PG Pilot	Mobile logbook	Upload	None
SkyViz	3D visualisation	Upload	None
IGC Viewer / BGA	Viewer / scoring	Upload	None
XCTrack	Instrument app	Records IGC	Platform API
Flyskyhy	Instrument app	Records IGC	Platform API

SeeYou Cloud is Naviter's cloud platform for flight analysis. Provides advanced IGC parsing and visualisation but operates as a closed ecosystem.

SkyVarg[5] is an AI-powered paragliding analytics platform offering thermal detection, XC route analysis, and flight scoring using machine learning.

PG Pilot[6] is a mobile-first paragliding logbook and social platform.

SkyViz[7] is a web-based 3D flight visualisation tool.

IGC Viewer / BGA Ladder[8] is the British Gliding Association's web-based IGC track viewer.

FAI/CIVL IGC file format [1]. The IGC file format specification defines the data format that igc-net exchanges. igc-net does not modify or extend the IGC format; it treats IGC files as opaque byte sequences identified by their BLAKE3 hash.

12 Conclusion

The soaring community generates a substantial and growing corpus of flight data, but the infrastructure for exchanging this data remains fragmented across isolated platform silos. igc-net addresses this structural problem by providing an open, peer-to-peer protocol that gives every IGC file a universal, content-addressed identity and makes it discoverable and fetchable by any participant.

The protocol is deliberately minimal. It gives platforms a neutral interoperability layer, and it gives pilots a way to place flights into a network archive that can outlive any single service. If archival nodes retain a flight, it remains retrievable as long as at least one such node continues to serve it.

The Rust reference implementation shows that the design is practical, embeddable, and testable. NTNU DSE Lab, part of NTNU—a public university and non-profit research environment—is continuing work on university-hosted archival capacity for igc-net.

We invite platform operators, tooling developers, federations, researchers, and pilots to review the specification, test the implementation, and help turn igc-net into a durable open archive and exchange layer for soaring flight data.

Table 12: Canonical protocol identifiers.

Identifier	Derivation	Canonical hex
Announce topic	`BLAKE3("igc-net/announce/v1")`	`2f0656...dda9`
Analytics topic	`BLAKE3("igc-net/analytics/v1")`	`4f2934...0950`
Metadata schema	literal	`igc-net/metadata`
Schema version	literal	`1`
Analytics prefix	literal	`igc-net/meta-doc/v1`

Table 13: Required base metadata fields.

Field	JSON type	Constraints
`schema`	string	Must equal `"igc-net/metadata"`
`schema_version`	integer	Must equal `1`
`igc_hash`	string	64-char lowercase BLAKE3 hex

A Canonical Protocol Identifiers

B Full Metadata Field Reference

Required fields.

Optional fields.

C Announcement Wire Format Examples

Minimal metadata blob.


```
{
  "schema": "igc-net/metadata",
  "schema_version": 1,
  "igc_hash": "3a7f2c8e1b4d5f6a9b0c...5f6a7b8c9d0e1f2a"
}
```


Announcement message.


```
{
  "igc_hash": "3a7f2c8e1b...0e1f2a",
  "meta_hash": "1b2c3d4e5f...0a1b2c",
  "node_id": "a1b2c3d4e5...0a1b2",
  "igc_ticket": "<BlobTicket encoding hash + node address>",
  "meta_ticket": "<BlobTicket encoding hash + node address>"
}
```


D Analytics Type Catalog

The following example illustrates the standard analytics blob format. Full examples for all standard analytics types (classification, site match, thermals, wind, segments) are provided in the protocol specification (`specs_meta.md`, Sections 15.1–15.5).

[5] https://skyvarg.ai
[6] https://pgpilot.app
[7] https://skyviz.io
[8] https://igcviewer.bgaladder.net

Table 14: Optional base metadata fields.

Field	Type	Source	Constraints
`original_filename`	string	Publisher	Upload-time filename
`flight_date`	string	`HFDTE`	`YYYY-MM-DD`
`started_at`	string	First B-record	`YYYY-MM-DDTHH:MM:SSZ`
`ended_at`	string	Last B-record	`YYYY-MM-DDTHH:MM:SSZ`
`duration_s`	integer	Derived	Non-negative seconds
`pilot_name`	string	`HFPLT`	As present in IGC
`glider_type`	string	`HFGTY`	As present in IGC
`glider_id`	string	`HFGID`	As present in IGC
`device_id`	string	A-record	Logger/device ID
`fix_count`	integer	Derived	Count of B-records
`valid_fix_count`	integer	Derived	Validity `A`
`bbox`	object	Derived	min/max lat/lon
`launch_lat`	number	First B-record	WGS84 degrees
`launch_lon`	number	First B-record	WGS84 degrees
`landing_lat`	number	Last B-record	WGS84 degrees
`landing_lon`	number	Last B-record	WGS84 degrees
`max_alt_m`	integer	Derived	Pressure alt. preferred
`min_alt_m`	integer	Derived	Pressure alt. preferred
`publisher_node_id`	string	Publisher	64-char Ed25519 hex
`published_at`	string	Publisher	`YYYY-MM-DDTHH:MM:SSZ`

Thermals (`igc-net/thermals/v1`).


```
{
  "schema": "igc-net/thermals",
  "schema_version": 1,
  "igc_hash": "<blake3-hex>",
  "computed_at": "2026-04-03T10:00:00Z",
  "thermals": [
    {
      "start_time": "2026-04-03T10:15:00Z",
      "end_time": "2026-04-03T10:18:30Z",
      "duration_s": 210,
      "centre_lat": 46.1234,
      "centre_lon": 7.5678,
      "radius_m": 45,
      "altitude_entry_m": 1200,
      "altitude_exit_m": 1850,
      "altitude_gain_m": 650,
      "climb_rate_ms": 3.1
    }
  ]
}
```


References

[1] FAI/IGC, "Technical Specification for IGC-approved GNSS Flight Recorders," International Gliding Commission, 2023.

[2] n0.computer, "Iroh: A toolkit for building distributed apps," https://docs.iroh.computer, 2024–2026.

[3] J. O'Connor, J.-P. Aumasson, S. Neves, and Z. Wilcox-O'Hearn, "BLAKE3: One function, fast everywhere," https://github.com/BLAKE3-team/BLAKE3-specs, 2020.

[4] J. Benet, "IPFS – Content Addressed, Versioned, P2P File System," arXiv:1407.3561, 2014.

[5] D. J. Bernstein, N. Duif, T. Lange, P. Schwabe, and B.-Y. Yang, "High-speed high-security signatures," *Journal of Cryptographic Engineering*, vol. 2, no. 2, pp. 77–89, 2012.

[6] J. Leitão, J. Pereira, and L. Rodrigues, "HyParView: A Membership Protocol for Reliable Gossip-Based Broadcast," in *Proc. IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, pp. 419–428, 2007.

[7] S. Bradner, "Key words for use in RFCs to Indicate Requirement Levels," RFC 2119, IETF, 1997.

[8] S. Josefsson and I. Liusvaara, "Edwards-Curve Digital Signature Algorithm (EdDSA)," RFC 8032, IETF, 2017.

[9] J. Iyengar and M. Thomson, "QUIC: A UDP-Based Multiplexed and Secure Transport," RFC 9000, IETF, 2021.

[10] XContest, "World Paragliding Rankings and Statistics," https://www.xcontest.org/world/en/, accessed April 2026.

[11] OLC, "Online Contest – Pair and Worldwide Gliding Competition," https://www.onlinecontest.org, accessed April 2026.