

AINETNET Genesis: Design and Single-Host Evaluation of Signed Object Exchange

Two Brain

Technical Report TB-AINET-TR-001 · Revision 3 · 31 October 2026 · Not peer reviewed

Abstract

AINETNET Genesis is an experimental application protocol for exchanging signed information objects over existing Internet transports. We describe version 0.1 and an exploratory evaluation on one Linux host. The design binds content, author identity, provenance claims, and a public/private policy to an immutable object identifier; it separates the author of an object from a node supplying a copy. Experiments assess retrieval cost, admission under concurrent load, manually republished copies, process-crash recovery, publication controls, and object-level interoperability between Go and Python. Six payloads were retrieved successfully in 20 measured repetitions each. Manual propagation preserved the signed record across 39 transfers. Eight simultaneous requests exposed a four-operation admission limit: 48 of 96 requests failed. In a post hoc queue experiment, two and four client workers each completed 96 of 96 jobs, whereas eight workers completed 12 of 96. A small catalog application produced the same arithmetic result through AINET and a local HTTP control. These observations establish limited implementation behavior, not superiority over the Web, Internet-scale resilience, or AI semantic interoperability. The report identifies admission control, independent implementations, and multi-host evaluation as the next empirical tasks.

1 Introduction

Software that retrieves information from several sources must distinguish the identity of a source, the bytes it received, and the claims made about those bytes. A transport connection can authenticate a peer without establishing the original authorship of a copied document. Conversely, an author signature can remain verifiable after the original host stops serving a document, but it does not identify an available replacement host. Treating these as separate questions is useful when designing data exchange for automated clients.

AINETNET Genesis studies a particular combination of these mechanisms: an immutable signed object, explicit local publication, and direct exchange between authenticated peers. Its intended future clients include AI systems, but version 0.1 contains no model inference. The present research question is narrower: can this implementation retain object identity and author verification across copies, while making the conditions and failure modes of retrieval observable? A second question concerns whether an application can select, retrieve, and cite these objects without relying on a mandatory central registration service.

The Internet and the Web already support machine-readable data, application protocols, signatures, mirrors, and distributed services. There is no single central server for the Web as a whole. This work therefore does not assume that these systems are deficient or incapable of the proposed functions. AINET is an application-layer experiment over IP, not a replacement for IP or physical infrastructure. Its current contribution is a small executable design and an auditable set of local observations. Establishing novelty or the need for a separate standard requires a broader review and comparisons with systems offering equivalent guarantees.

2 System model and design

2.1 Identities, objects, and providers

Each node persists an Ed25519 key pair and derives an AINET identifier from its public key. The implementation uses the same key to establish the corresponding libp2p peer identity. Peers are registered manually; the claimed AINET identity is checked against the key authenticated by the connection. This authenticates possession of a key. It does not certify a person's name, an institutional affiliation, or an author's legal rights.

An object comprises a payload and a signed record. The record contains a manifest, an object identifier, and an Ed25519 signature. The manifest includes the payload's SHA-256 digest, media type, exact byte length, author identifier and public key, creation time, provenance claims, references, and access policy. An object identifier hashes the deterministically encoded manifest with a domain-separation prefix. It is consequently different from a hash of

the payload alone: changing the author, timestamp, or policy changes the identifier even when the payload is unchanged. The signature binds the author key to the object identifier through a separate domain-separated input. The exact encoding and prefixes are specified in the accompanying wire specification.

Verification checks the author/key relation, re-encodes the manifest, recomputes the identifier, verifies the signature, and checks the payload’s size, hash, and format. Text, PNG, and JPEG are supported by the implementation; the performance experiment used text and PNG. The provider authenticated on a live connection can differ from the author embedded in the record. Copies can therefore be checked without requiring the original author to remain online. This property says nothing about whether the author’s claims are true.

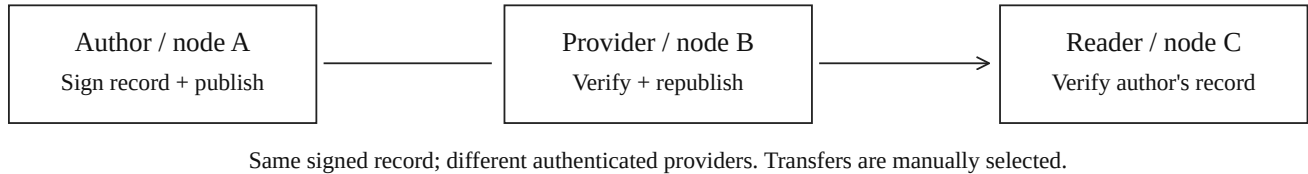


Figure 1. Separation of author and provider. Copies are selected and republished manually; the schematic does not imply automatic discovery.

2.2 Transport and publication

Nodes use go-libp2p with TCP, Noise, and Yamux. Control frames contain a bounded, deterministically encoded CBOR envelope; object payload bytes follow the object record without Base64 wrapping. The profile restricts the accepted CBOR types and rejects duplicate keys, noncanonical encodings, and unknown fields. It uses existing cryptographic libraries rather than a new cryptographic primitive. The prototype implements connection negotiation, object retrieval, and structured messages. Discovery and distributed query remain outside the evaluated implementation.

New local objects are private by default. Making a private object public creates a signed public version with a different identifier. A receiving node does not automatically serve its copy; local republication must be explicit. Stopping publication prevents subsequent retrieval from that node but cannot recall copies already distributed by another node. Thus the signed policy and the provider’s current serving decision are distinct. The two-state policy is not a general authorization system and does not implement recipient-specific encryption or revocation.

Storage uses node-specific local directories. Incoming bytes are staged, checked, and committed before successful completion is acknowledged. Keys depend on operating-system file permissions, and the local operating-system user and administrator are trusted. Limits bound message and object sizes, transfer duration, and concurrent work. These are implementation controls, not evidence of comprehensive resistance to hostile peers. Remote code execution and interpretation of received prose as system instructions are absent from the prototype.

3 Experimental method

All observations were collected on 31 October 2026 using one Linux x86-64 host with an Intel Xeon Platinum 8573C processor. The process environment reported five available CPUs; cgroup limits were four CPU cores’ worth of execution time and 16 GiB of memory. Nodes were independent processes with separate keys and storage, communicating over 127.0.0.1. These are process-level experiments, not independent machines or geographic sites. The Go executable was fixed across the original and follow-up studies. Synthetic inputs used seed 20261031; runtime keys were generated independently and were not retained in the distribution.

The original plan defined seven experiments, R1–R7. R1 retrieved three text payloads (1 KiB, 64 KiB, and 1 MiB) and three random-RGB PNGs (512, 1024, and 2048 pixels per side). Each fixture had one first retrieval, two warm-up retrievals, and 20 measured retrievals; fixture order was randomized within repetitions. R2 submitted one, two, four, or eight simultaneous requests to the same receiving node, using 1 MiB text and an approximately 3 MiB PNG, with six batches per condition. All R1 requests (including first and warm-up retrievals) and R2 requests total 318 recorded rows. Recovery checks after R2 batches were recorded separately.

R3 propagated a 64 KiB text object through manually configured chains of three, five, and eight nodes, with three runs per chain length. The script explicitly fetched, verified, and republished each copy before stopping the preceding provider. R4 used SIGKILL after acknowledged storage/publication in five trials and after observing a

pending receive file in five further trials. R5 repeated six publication or mutation checks on each of 20 synthetic texts. R6 tested object-level cross-language verification and malformed inputs. R7 built a signed catalog and a deterministic data-processing application, together with a local HTTP control. After R2 exposed rejection under load, R8 added a client queue: 32 jobs per batch, two/four/eight workers, three batches per condition, and 288 requests. R8 was post hoc; it did not modify the Go server or retry failed requests.

Retrieval latency measures the complete CLI operation, including process startup, local IPC, HELLO exchange, provider-side verification, transfer, receiver-side verification, and storage. It is not network transit time. The same objects and daemon processes were reused, and OS caches were not cleared. A first retrieval means absence at the receiving node, not a cold disk or cold OS cache. Medians, quartiles, and interpolated empirical p95 values describe the recorded sample; no confidence intervals or population-level guarantees are inferred. Requests within a batch and repetitions sharing a host are not independent deployment samples. A pilot run was excluded from reported results.

4 Results

4.1 Retrieval cost

All 120 measured R1 retrievals succeeded. Table 1 gives the observed distributions. Median completion time rose from 11.6 ms for 1 KiB text to 375.0 ms for the largest PNG. The 512-pixel PNG had a maximum of 391.6 ms despite a median of 39.4 ms. That observation was retained; the experiment did not identify its cause. Random-RGB PNGs compress poorly and are not representative of typical photographs or diagrams. Media decoding also contributes to validation, so payload size alone does not explain the time difference between text and images.

Table 1. R1 retrieval time (ms). Twenty measured requests per fixture; all succeeded.

Fixture	Bytes	Median	Q1–Q3	p95	Max.
Text 1 KiB	1,024	11.6	10.4–15.1	17.1	25.8
Text 64 KiB	65,536	13.2	12.5–14.2	19.5	20.0
Text 1 MiB	1,048,576	19.5	17.4–21.3	28.0	28.2
PNG 512 x 512	788,087	39.4	33.5–44.4	77.7	391.6
PNG 1024 x 1024	3,151,227	118.9	113.3–131.8	142.2	154.6
PNG 2048 x 2048	12,598,409	375.0	366.0–414.2	492.7	565.3

CPU counters covered the two node daemons, excluding the CLI and Python harness. Their resolution produced a median of 0 ms for small text operations; this is not evidence of zero computation. Combined daemon RSS was sampled approximately every 10 ms. The largest PNG trials reached a maximum observed 197.7 MiB. Because daemons were reused, retained allocations and previous large inputs affect later samples. RSS therefore cannot be interpreted as the incremental memory required by one payload, and sampling can miss shorter peaks. Function-level profiling was not performed, so attributing a specific proportion of CPU cost to decoding, hashing, or storage would be premature.

4.2 Concurrent admission and queued submission

R2 completed 12/12 requests at concurrency one, 24/24 at two, and 48/48 at four. At concurrency eight, 48 of 96 requests succeeded. Of the 48 failures, 47 returned a busy error and one returned a HELLO authentication/version error. The latter diagnostic conflates more than one failure path; the observation cannot establish a bad key or its precise root cause. All 48 post-batch single-request recovery checks succeeded. The implementation shares four processing slots across relevant incoming and outgoing work, so rejecting additional requests is consistent with its admission mechanism.

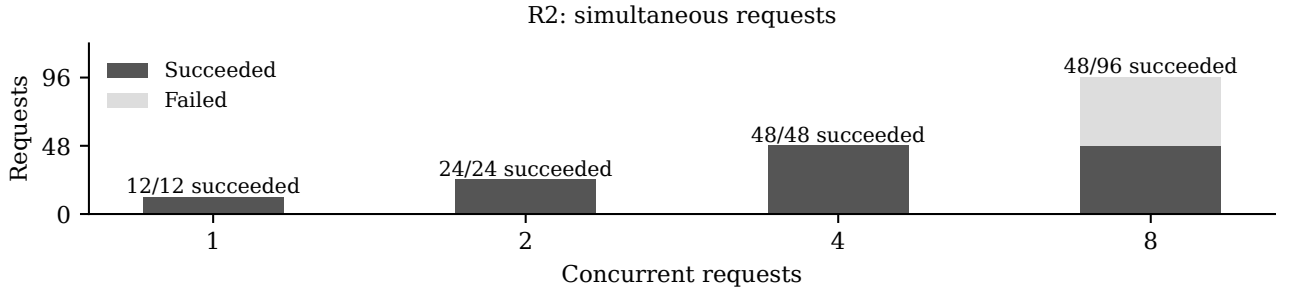


Figure 2. R2 successes and failures. Two payloads and six batches per condition; the request count varies with concurrency.

In R8, the two- and four-worker queues completed all 96 jobs per condition. The eight-worker queue completed only 12/96. When active jobs occupy the available slots, quickly rejected requests free client workers to submit further jobs, which can also be rejected. This accounts for the rejection pattern without implying that the server successfully processed the batch faster. Table 2 includes time from batch submission, including client-side waiting, for successful jobs. The 0.292 s median at eight workers is conditional on a small surviving subset and is not a useful speed advantage. With only three batches per condition, the data do not justify declaring either two or four workers universally optimal.

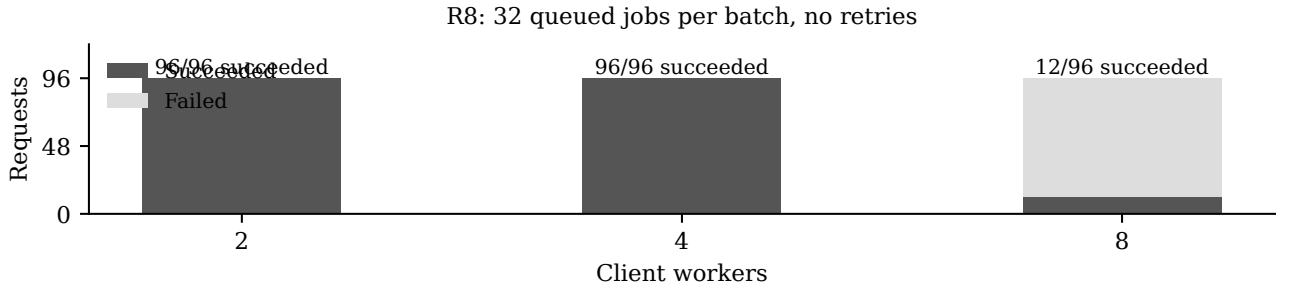


Figure 3. R8 successes and failures: ~3 MiB PNG, three batches of 32 jobs, no retries. This is a different workload from Figure 2, not a before/after comparison.

Table 2. R8 successes and successful-job completion times including queueing (s).

Workers	Successes/requests	Median	p95
2	96/96	1.385	2.489
4	96/96	1.429	2.431
8	12/96	0.292	0.385

4.3 Copies, recovery, and verification

R3 preserved byte-for-byte equality of the complete signed record over 6, 12, and 21 transfers for the three-, five-, and eight-node chains respectively, totaling 39. In all nine runs the final node verified the object after preceding providers had stopped. Nine direct attempts against stopped providers failed as expected. This demonstrates validation of manually supplied copies, not automatic replica discovery, failover routing, or tolerance of a host-level failure.

All five post-acknowledgement R4 trials recovered the same identity, verified stored objects, and retained publication state. In the five in-flight trials a pending file was observed before SIGKILL, no committed record was present before restart, and restart cleanup followed by retrieval succeeded. Observation and termination were not simultaneous, so the exact last executed instruction is unknown. SIGKILL leaves the kernel and filesystem running; these results do not test power loss, storage-device failure, every commit boundary, or disk exhaustion. A crash after blob placement but before metadata commitment can leave an unreferenced blob, motivating later garbage collection.

Table 3. Selected functional observations. Rows use different units and are not additive.

Study	Count	Observation
R3	39/39	Signed-record equality across transfers
R4	5 + 5	Recovery after acknowledgement / during receive
R5	120/120	Publication and local-mutation checks
R6	6 + 20	Go→Python / Python→Go
R6	64 + 64 + 6	Payload / signature mutations / malformed CBOR rejected

R5’s 120 checks cover six repeated behaviors: indistinguishable generic errors for private/missing objects, retrieval after explicit publication, no automatic serving of received copies, rejection after publication is stopped, and rejection of locally modified payloads or signatures. Matching error text does not prove absence of timing or traffic-analysis leakage. Mutation cases inject faults into a controlled local store; they are not remote attack demonstrations.

R6’s Python verifier uses a separate minimal CBOR implementation and an existing Ed25519 library. Six Go-produced fixtures verified in Python, and 20 Python-produced text objects verified in Go. The Python verifier rejected 64 payload mutations, 64 signature mutations, and six malformed CBOR cases. The Python-generated objects used a public RFC 8032 test key, suitable only for test vectors. This is object-layer interoperability, not a second complete Noise/libp2p implementation. Both implementations were produced with the same development assistance, so the result does not substitute for independent authorship or review. Overall, the original study recorded 412 successful checks out of 412; repeated checks and measurement requests are different accounting units.

4.4 Catalog processing and the HTTP control

R7 represented a catalog as JSON inside a text object. The application selected four synthetic temperatures, 18, 20, 22, and 24 degrees Celsius, fetched their objects, checked types and units, and returned a mean of 21 with supporting object identifiers in a structured answer. A private synthetic document was omitted from the catalog. These links belong to application-level JSON and do not constitute a new manifest-reference API or an implementation of protocol-level DISCOVER or QUERY. The initial catalog identifier and provider were known in advance.

The local HTTP control exposed the same catalog and values and returned the same mean. When 18 was changed to 19, the control could replace the representation at the same path, while AINET assigned a new identifier and retained the old object’s value. The Web can equally provide immutable or versioned URLs, hashes, and signed representations. The control therefore illustrates application behavior rather than a fundamental Web limitation. It did not match HTTPS, signatures, caching, or persistence guarantees, and no performance ranking is drawn. Computation was fixed arithmetic; neither model-generated answers nor semantic understanding were evaluated.

5 Interpretation and relation to existing systems

The experiments separate three properties often conflated in descriptions of decentralized information: verification of a specific object, availability of a provider, and agreement about the meaning of its content. R3 offers limited evidence for the first property across manually chosen providers. It does not address global availability or agreement. R7 shows that object identifiers can be carried into an answer as explicit references, but a correct reference is not evidence that an AI interpreted it correctly. The cryptographic and application layers must remain distinct in subsequent evaluations.

HTTP supports signed representations, structured data, access control, immutable references, and replicas at the application layer [1]. Deterministic CBOR [2] and Ed25519 [3] are existing building blocks. Neither their combination nor the preservation of an author signature across copies is a new security property. R3 is an integration check of this implementation. The current study does not establish that a custom protocol is necessary.

IPFS uses content identifiers [4], while IPNS provides signed, mutable references that can be validated independently of their distribution path [7]. NDN explicitly separates trust in data from trust in its forwarding path [8]. Nostr NIP-01 describes author-signed events distributed by relays, including replaceable and addressable event

rules [9]. These mechanisms overlap with AINET’s author/provider separation; AINET does not implement their discovery, naming, or update facilities. Differences in key algorithm or serialization are not, on their own, practical advantages.

COSE Hash Envelope provides a standardized way to carry a payload hash in COSE and is a candidate replacement for custom signature packaging [10]. A matching profile would still need to define metadata, publication decisions, and receipt semantics. AINET’s integer-key CBOR and rejection of tags differ from DAG-CBOR’s string-key maps and CID links [11]. Its identifiers are not interoperable CIDs. The initial compact, fixed-schema choice has an interoperability cost that this study did not measure. C2PA provides a broader framework for signed media-provenance claims [12]; AINET’s free-text Provenance fields are signed assertions, not independently checked editing history. RDF likewise addresses relationship representation rather than solving this entire trust model [5].

The next comparison must match signatures, verification, persistence, caching, trust inputs, and resource budgets. Transport and packaging should be varied separately: first the same object over AINET and HTTPS, then equivalent metadata under the custom and COSE packaging. If a standard approach meets the requirements without a measurable benefit from the custom profile, adopting that standard is the preferred outcome. The accompanying study 002 plan is a draft, not an executed or externally preregistered experiment.

The clearest engineering implication is that success-only latency is an incomplete measure for a bounded service. Admission rejects work rather than delaying it, and a client queue can change the observed completion fraction without changing server code. Subsequent evaluation should measure useful completed work, waiting-time distributions, cancellation, bounded retries, and fairness between peers. A two-worker default is a conservative candidate for the next adapter, not a finding about optimal concurrency on arbitrary hardware.

Several threats to validity remain. All nodes shared a host, scheduler, memory budget, and filesystem environment. The small number of repetitions, cache reuse, fixed execution history, and post hoc queue experiment limit causal and statistical claims. Synthetic text and random images do not represent a deployment workload. Publication controls depend on trusted local administration and do not revoke remote copies. No external security audit, Mac execution, cross-machine transfer, Internet-scale discovery, independent full-protocol implementation, or AI-model evaluation was performed in this study.

6 Conclusion and reproducibility

Version 0.1 supports a locally demonstrated path from explicitly published information to verified retrieval, manual republication, and reference-bearing application results. The same evaluation exposes rejection under concurrent demand and important limits on claims about availability and trust. The next steps are controlled multi-host trials, a separately authored interoperable peer, a queued retrieval interface, and storage-capacity management. Model integration belongs to version 0.2 and should test explicit input permissions, citation correctness, and task-level meaning against defined ground truth.

The archived evidence includes CSV/JSONL request records, check results, event logs, environment metadata, fixture hashes, experimental plans, and object test vectors. CSV and JSONL encode the same requests and must not be counted twice. The original 318 requests and the follow-up 288 requests total 606, including failures and warm-up traffic; this is not a 606/606 success claim. Source and executable hashes identify the measured artifact. A rebuild or a different environment requires a new run and a new interpretation rather than reuse of this report’s conclusions. The English and Japanese editions describe the same archived experiments. Revision 3 changes interpretation and specification detail without replacing those measurements. The baseline source is preserved under `research/security/baseline-source/`. A separate later security study, TB-AINET-SEC-001, repairs admission, receive locking, and cancellation; its results must not be attributed to this measured executable. The preview provides measurements and documents, but a complete public source release and distribution license remain pending. The local plan hash and any self-signed tag are not independent timestamp evidence. No independent human review is claimed.

Research record. Two Brain, AINTERNET Genesis, technical report TB-AINET-TR-001, revision 3. Exploratory technical report; not peer reviewed. Implementation, experiments, analysis, and manuscript preparation were assisted by Codex. No independent audit or third-party replication is claimed. The measured executable SHA-256 is 872cbb9fa7698e0cefa94cb70f9d7ba6d66457392a14dae179a345477c194a36.

References

- [1] R. Fielding, M. Nottingham, and J. Reschke. *HTTP Semantics*. RFC 9110, 2022. <https://www.rfc-editor.org/rfc/rfc9110>.
- [2] C. Bormann and P. Hoffman. *Concise Binary Object Representation (CBOR)*. RFC 8949, 2020, §4.2.1. <https://www.rfc-editor.org/rfc/rfc8949>.
- [3] S. Josefsson and I. Liusvaara. *Edwards-Curve Digital Signature Algorithm (EdDSA)*. RFC 8032, 2017. <https://www.rfc-editor.org/rfc/rfc8032>.
- [4] IPFS Documentation. *Content Identifiers (CIDs)*. <https://docs.ipfs.tech/concepts/content-addressing/>. Accessed 31 October 2026.
- [5] R. Cyganiak, D. Wood, and M. Lanthaler, editors. *RDF 1.1 Concepts and Abstract Syntax*. W3C Recommendation, 2014. <https://www.w3.org/TR/rdf11-concepts/>.
- [6] Two Brain. *AINET Language / ID / Object Wire Specification 1*; experiment plans and archived results, AINTERNET Genesis 0.1, 2026. Accompanying project artifacts: `spec/wire.md`, `research/PLAN.md`, `research/FOLLOWUP.md`, and `research/results/`.
- [7] IPFS Documentation. *InterPlanetary Name System (IPNS)*. <https://docs.ipfs.tech/concepts/ipns/>.
- [8] Named Data Networking Project. *Architectural Overview*. <https://named-data.net/project/archoverview/>.
- [9] Nostr. *NIP-01: Basic protocol flow description*. <https://github.com/nostr-protocol/nips/blob/master/01.md>.
- [10] O. Steele, S. Lasker, and H. Birkholz. *COSE Hash Envelope*. RFC 9995, 2026. <https://www.rfc-editor.org/rfc/rfc9995.html>.
- [11] IPLD. *DAG-CBOR Specification*. <https://ipld.io/specs/codecs/dag-cbor/spec/>.
- [12] C2PA. *Technical Specification*, version 2.2. https://spec.c2pa.org/specifications/specifications/2.2/specs/C2PA_Specification.html. Accessed 31 October 2026.