

Controlled Adversarial Testing and Availability Repairs in AINET 0.1

Two Brain

Technical Report TB-AINET-SEC-001 · Revision 1 · 31 October 2026 · Not peer reviewed

Abstract

AINET 0.1 is an experimental protocol for exchanging signed objects over existing IP networks. Its use of authenticated transport and object signatures does not, by itself, establish that a peer will make progress or share resources fairly. We therefore evaluated the implementation against controlled malformed and stalled peers on one Linux host. A separate adversarial process exposed two availability defects: one registered peer could occupy every application slot, and a stalled object receive held the store-wide mutex. A separate Go API experiment showed that a caller’s deadline did not interrupt protocol reads after stream establishment. We repaired admission accounting, staged network input outside the store lock with bounded disk reservations, and propagated cancellation to established streams. The identical process-level harness passed 70 of 80 checks before the repairs and 80 of 80 afterward. Honest retrieval completed within a two-second observation deadline in 0/5 versus 5/5 trials for each availability condition. These observations establish specific implementation failures and their local regression results. They do not establish a new cryptographic mechanism, resistance to distributed denial of service, or an advantage over HTTPS, IPFS, or Nostr.

1 Motivation and research question

Two Brain’s longer-term aim is to investigate how software and AI agents can exchange information with explicit identity, provenance claims, and publication decisions. Cyberattacks motivate examining the boundaries of such a system, but their existence does not demonstrate that IP or the Web is obsolete. The Web supports structured data, application-level signatures, authorization, and replicated distribution. AINET currently reuses IP, standard cryptography, and go-libp2p. Its security must be assessed at the boundaries it adds, rather than inferred from a different name or data format.

The preceding exploratory report, TB-AINET-TR-001, established local object exchange and documented overload rejection [1]. Those observations were principally integration and fault-injection checks; the preservation of a valid signature across copies is an established property. This follow-up asks a narrower, falsifiable question: **can a registered peer that remains cryptographically valid but stops making progress prevent an unrelated, honest retrieval from completing?** We also check whether invalid data becomes committed, whether uncertain message receipt produces duplicate storage, and whether a caller can terminate an established operation. The contribution is a reproducible diagnosis and repair of this implementation, not a claim that its custom protocol is necessary.

2 Scope, adversaries, and method

The target is AINET 0.1 using TCP, Noise, Yamux, Ed25519, SHA-256, and a restricted deterministic CBOR profile. Publication requires both a signed public policy and an explicit local serving decision. Peers are registered manually, and the implementation permits literal loopback addresses only. There is no automatic discovery, remote execution, or AI model in this study.

Table 1. Threat model and evaluated boundaries.

Actor or resource	Treatment
Unregistered peer	Check registration and key binding; reject admission
Registered malicious provider	Mutate records, payloads, correlation, and lengths
Registered stalled peer	Withhold requests or payload; measure an unrelated honest job
OS, administrator, harness	Trusted; OS compromise and key theft not tested
Network and data	One host, 127.0.0.1, synthetic text

The process-level lab starts a victim node, an honest reader, and a programmable probe as separate Linux processes with separate keys and storage. It also starts an unregistered node for one admission check. The probe uses the real transport and authenticated peer registration; it can then send malformed frames, substitute records, withhold bytes, or withhold application progress. Only synthetic objects and author-controlled 127.0.0.1 endpoints are used. No external service is targeted. Temporary node directories and private keys are removed at teardown.

The same probe executable, Python harness, local plan, and synthetic inputs were used for the baseline and repaired node executables. The baseline completed at 12:49:18 UTC and the repaired run at 12:54:38 UTC on 31 October 2026. Both environments report Linux 6.18.57 x86-64 with glibc 2.41. This is a sequential before/after diagnostic study on a shared host, without randomization or independent deployment samples. The original binary and source snapshot remain archived. The repaired binary was built with Go 1.26.10; full hashes and raw observations identify the artifacts.

Malformed-request and provider conditions use three repetitions where applicable. Availability conditions use five repetitions each. The honest job is retrieval of a small public text object; success requires the CLI operation to finish within two seconds. This threshold is a local test criterion, not a production service-level objective. An early rejection counts as a failure of the honest job, not fast useful service. The harness checks the attack's setup before measuring the job and records recovery after releasing the stalled streams. Each JSONL row is one assertion, sometimes repeated. Eighty assertions are not eighty independent security properties, nor eighty retrieval requests.

The plan was written locally before these executions. Its hash records which plan was used; it is not an externally timestamped preregistration. The API deadline test runs real libp2p connections within one Go test process and is reported separately. It sets a 250 ms caller deadline, allows at most 800 ms for observed termination, and makes the remote peer remain silent for 1.2 s. HELLO and post-request response stalls are tested five times each. A later explicit-cancel case and a near-quota receive regression supplement the matched experiments.

3 Reproduced defects and repairs

3.1 One peer occupied every processing slot

The baseline allocated one of four shared application slots to each accepted inbound stream. There was no per-peer application limit. A registered probe completed HELLO on four streams and then withheld each request. All four slots remained occupied. An unrelated reader's five retrievals failed immediately, reporting a combined HELLO authentication/version diagnostic. In this setup the failure followed exhausted admission capacity; the error wording alone would not establish an invalid identity.

The repair retains the global limit of four and admits at most two inbound operations for one authenticated peer. Admission and per-peer counters are updated under the node mutex, with corresponding release on completion. In the repeated experiment, two of the probe's four streams were accepted and two were rejected; all five honest jobs completed within the observation deadline. This limits one peer's occupancy. Two colluding registered peers, heavy outgoing work, or other shared bottlenecks can still exhaust capacity. There is no waiting queue, proportional fairness, or automatic retry in this repair; exceeding the limit resets a stream. The reserved BUSY response remains unimplemented.

3.2 A stalled receive blocked unrelated stored objects

The baseline received an object's payload while holding the store-wide mutex. A provider could send a valid signed record and then withhold its payload. While the victim waited for bytes, another peer's request for a previously stored public object could not enter the storage operation. All five honest jobs exceeded the two-second observation limit. The experiment released the stalled provider afterward; it does not show an indefinitely stuck operation beyond the implementation's own transfer timeout.

The repair divides receive into admission, staging, and commitment. It validates metadata and reserves a bounded payload-plus-metadata allowance under the mutex, then receives and validates the temporary file outside that lock. It reacquires the lock, releases its reservation, rechecks count and capacity constraints, and commits the verified object. Failure releases reservations and removes the pending file. Other in-flight reservations remain counted. Temporary bytes and reservations may be counted conservatively together, so capacity can be rejected before the physical quota is full. This favors avoiding overcommit over maximizing utilization.

All five unrelated retrievals subsequently completed within the deadline. A storage regression additionally uses a sparse file to bring logical usage near the quota, blocks one receive, confirms that listing still completes, rejects a second receive that would exceed the reserved allowance, and confirms cleanup and later valid receipt. This tests quota accounting and lock scope; it is not a physical disk-exhaustion or power-loss experiment. Local validation, filesystem synchronization, and some local file operations can still hold the mutex.

3.3 Caller cancellation stopped at stream establishment

The API passed its context to stream opening but subsequently applied only internal phase deadlines. A caller’s 250 ms deadline could expire while the established stream continued waiting. In the baseline’s ten subtrials, operations returned only when the test peer released the stream at approximately 1.2 s. These are observed waits terminated by the test, not a measurement of the longest possible hold.

The repair binds context cancellation to a stream reset and uses the earlier of the caller deadline and the internal phase deadline. Read and write errors retain the context error when cancellation has occurred; cleanup removes the callback. The ten repeated subtrials then terminated at approximately 250–251 ms, below the 800 ms test limit. An additional cancellation-without-deadline regression returned promptly under the race detector. This repair concerns Go API network I/O. It does not claim to interrupt arbitrary CPU work, a blocked filesystem, or a CLI command merely because its terminal disconnects.

4 Results and supporting checks

Table 2. Before/after observations. Rows have different units and are not additive.

Check or job	Baseline	Repaired
Passing process-level assertions	70/80	80/80
S3 honest retrieval ≤ 2 s	0/5	5/5
S4 honest retrieval ≤ 2 s	0/5	5/5
S2 provider rejection and cleanup	24/24	24/24
API deadline: termination ≤ 800 ms	0/10	10/10
API deadline: observed median	1204.68 ms	250.67 ms

The baseline’s ten failed process-level checks are exactly the five honest jobs in each availability condition. Both versions rejected the tested unregistered or falsely identified peers and malformed requests. Across eight provider conditions and three repetitions, both rejected invalid signatures, inconsistent author identity, an unrequested object, a mismatched response reference, an oversized control frame, altered content, truncated transfer, and a private object returned for retrieval. No new committed record or leftover temporary receive file was observed in those cases. These are selected concrete cases, not a measured detection rate for arbitrary adversarial traffic.

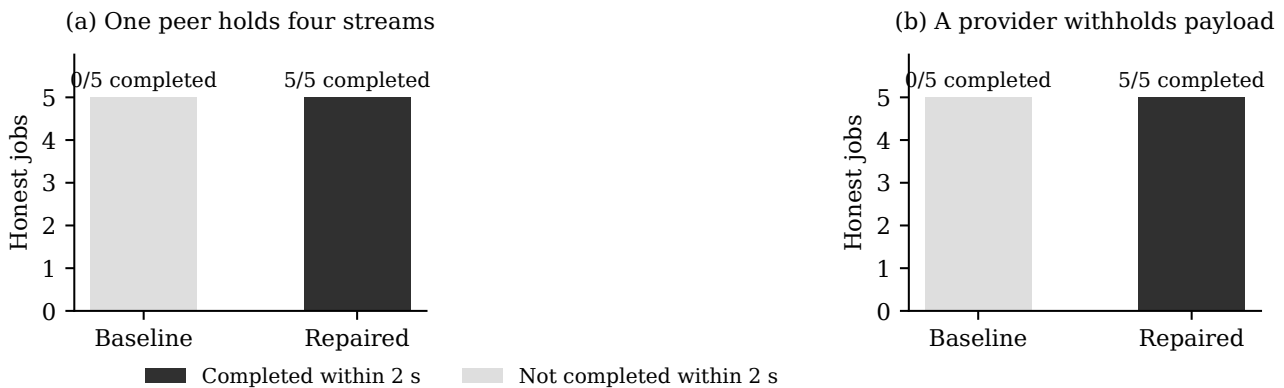


Figure 1. Honest retrieval under matched stall conditions; five trials per condition and a two-second observation deadline. Baseline jobs are rejected in (a) and time out in (b). Neither is a successful completion-time measurement.

The receipt experiment deliberately sends a MESSAGE without reading its receipt, observes that the recipient stored it, and then closes that stream. Resending the same sender/ID/content yields a receipt and one stored message; reusing that ID with different content is rejected. Three such groups and a restart check passed in both versions. This models an application uncertain about successful delivery, rather than injecting packet loss into a physical link. Deduplication persists while the local inbox survives; it is capped at 1,024 entries with no automatic expiration. The CLI does not expose stable-ID retries. Object retrieval similarly verifies a repeated object ID, but may rewrite its files and append verification events. Neither behavior provides exactly-once distributed execution.

The tested private key was mode 0600. The verification log did not contain the synthetic payload markers checked by the harness. Instruction-like text remained an object payload; no language model processed it, so this says nothing about model prompt-injection resistance. Ordinary CLI end-to-end regression passed 31 checks, including text, PNG, JPEG, publication, tampering, message exchange, and restart. Go race tests, vet, and module verification passed. A ten-second CBOR decoder fuzz run executed 108,311 inputs without an observed failure. The run budget and existing corpus limit that observation.

Govulncheck v1.9.0 reported zero reachable known vulnerabilities and zero additional vulnerable imported packages for this build. It also reported one advisory in a required module: GO-2026-5932, concerning the unused golang.org/x/crypto/openpgp package [2]. The application does not import that package. The advisory is retained in the results rather than described as a completely clean dependency inventory. Database-based checks cannot exclude undisclosed vulnerabilities.

```
+ uname -srm
Linux 6.18.57 x86_64
+ python3 research/security/summarize.py --show comparison
AINETNET 0.1 | controlled Linux security experiment
Independent processes, 127.0.0.1, synthetic data; no external targets.
Saved observations are displayed below. This is not a new test run.

BASELINE-01 completed 2026-10-31T12:49:18Z
Checks: 70 passed / 80; failed: 10
Honest retrieval while one peer holds streams
  Within 2 s: 0/5; successful-job median: n/a (no successful jobs)
Honest retrieval while another receive stalls
  Within 2 s: 0/5; successful-job median: n/a (no successful jobs)
Binary SHA-256: 872cbb9fa7698e0cefa94cb7...

FIXED-01 completed 2026-10-31T12:54:38Z
Checks: 80 passed / 80; failed: 0
Honest retrieval while one peer holds streams
  Within 2 s: 5/5; successful-job median: 11.41 ms
Honest retrieval while another receive stalls
  Within 2 s: 5/5; successful-job median: 12.32 ms
Binary SHA-256: dfe682b8688bd68a541d9f44...

Full observations: research/security/results/*/results.jsonl
Scope: these cases do not establish Internet-scale or AI-model security.
+ set +x
```

Figure 2. Actual Linux xterm displaying aggregation of saved observations. Baseline appears above repaired results, with deadline completion counts and successful-job medians. The capture documents result inspection, not the original traffic in progress.

5 Interpretation and remaining exposure

Authentication identifies a key holder; it does not ensure that the holder promptly completes a protocol. Signature verification detects inconsistent bytes and claims but cannot prevent a valid peer from consuming scarce time, streams, memory, or disk. The repaired defects arose in resource ownership, lock scope, and cancellation propagation. These implementation concerns also occur in HTTP and other peer-to-peer software; this study supplies no comparative evidence that AINET’s custom encoding or transport is safer.

Content addressing and verification independent of a live provider have substantial prior art. IPFS/IPNS, NDN, and Nostr address related data-identification or signed-distribution concerns [3–5]. COSE Hash Envelope is a candidate standard packaging mechanism [6]; adopting it would still require a defined metadata and publication contract. AINET’s integer-key CBOR is not DAG-CBOR, and its object identifiers are not interoperable CIDs [7]. The next

comparison should keep author signatures, trust inputs, verification, persistence, caching, and resource budgets constant while varying the transport or packaging independently. Choosing an existing standard remains a valid outcome.

Known boundaries remain significant. The experiment does not cover independent hosts, latency/loss injection, NAT, hostile operating-system users, stolen keys, publisher collusion, Sybil identities, sustained multi-peer load, or Internet-scale flooding. Manual admission limits who can reach the application but does not establish reputation. Public copies cannot be remotely recalled. Signatures do not certify truth, organizational affiliation, safe instructions, or freshness; the system has no latest-version resolver or key-revocation protocol. No power-loss guarantee is inferred from file synchronization or process restart checks. Human independent review and third-party replication have not been performed.

6 Reproduction and next experiments

The local repository contains the baseline source snapshot, measured executables, loopback-only probe, harness, regression tests, raw rows, hashes, logs, and terminal capture script. `research/security/README.md` gives commands and distinguishes rebuilding from reusing a measured binary. The public evidence download contains measurements, plans, verification logs, and screenshots. It is not a complete executable source release: the project distribution license and public source release remain pending. Artifact hashes identify bytes but do not establish independent authorship or time of creation. The terminal figures show actual Linux terminal windows displaying saved observations; they are not fabricated output or photographs taken during the original requests.

Priority follow-ups are a bounded admission queue with explicit overload signaling; deadline-aware retries and cancellation; sustained competing clients and multiple authorized hostile peers; injected delay, loss, and disconnection on separately controlled hosts; and an equal-guarantee HTTPS baseline. Those experiments must measure the fraction of all offered jobs completed by their deadline, alongside bytes, CPU/memory, and peer fairness. Formal comparison parameters should be frozen before collecting the comparison data. AI evaluation belongs to a later layer, where citation attribution and interpretation of malicious but correctly signed content need distinct ground-truth tests.

Research record. Two Brain, TB-AINET-SEC-001, revision 1, 31 October 2026. Engineering security study; not peer reviewed. Implementation, experiments, analysis, and manuscript preparation were assisted by Codex. This is manual source inspection and controlled local testing, not an external audit or a completed dedicated Codex Security scan. Study 001’s historical measurements are unchanged; this report evaluates different, explicitly identified repaired code.

References

- [1] Two Brain. *AINETNET Genesis: Design and Single-Host Evaluation of Signed Object Exchange*. TB-AINET-TR-001, revision 3, 2026. Accompanying report and archived measurements.
- [2] Go Vulnerability Database. *GO-2026-5932*. <https://pkg.go.dev/vuln/GO-2026-5932>. Tool observation: 31 October 2026.
- [3] IPFS Documentation. *InterPlanetary Name System (IPNS)*. <https://docs.ipfs.tech/concepts/ipns/>.
- [4] Named Data Networking Project. *Architectural Overview*. <https://named-data.net/project/archoverview/>.
- [5] Nostr. *NIP-01: Basic protocol flow description*. <https://github.com/nostr-protocol/nips/blob/master/01.md>.
- [6] O. Steele, S. Lasker, and H. Birkholz. *COSE Hash Envelope*. RFC 9995, 2026. <https://www.rfc-editor.org/rfc/rfc9995.html>.
- [7] IPLD. *DAG-CBOR Specification*. <https://ipld.io/specs/codecs/dag-cbor/spec/>. Online specifications accessed 31 October 2026.