Independent Submission
Internet-Draft
Intended status: Experimental
Expires: 26 September 2026

J. Song
HKUST
M. Yuan
CUHK

25 March 2026

Agent Name System (ANS)
draft-song-anp-ans-00

Abstract

This document defines the Agent Name System (ANS), a name registration and resolution protocol for autonomous AI agents in the Agent Network Protocol (ANP) suite. ANS maps agent:// URIs to network-layer peer identifiers, providing the binding between human-readable agent names and the cryptographic peer identities used by the Agent Internet Protocol (AIP) for datagram delivery.

ANS defines a Name Record format, four AITP method names for name operations (ans.register, ans.resolve, ans.unregister, ans.lookup), a multi-layer resolution algorithm, and two dissemination mechanisms (GossipSub announcements and DHT storage). ANS supports three addressing modes — unicast, anycast, and channel — over a single URI syntax.

ANS is intentionally a narrow name-binding layer: it maps names to peers and tags, but defers capability description to the Agent Description Protocol (ADP), reputation and ranking to companion protocols, and economic anti-spam mechanisms to deployment profiles.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at https://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on 26 September 2026.

Table of Contents

1. Introduction

1.1. Problem Statement

Agents in a decentralized network need stable, human-readable names that can be resolved to the cryptographic peer identifiers used for datagram delivery. The Agent Internet Protocol (AIP) [I-D.song-anp-aip] defines agent:// URIs as the addressing scheme for datagrams, but AIP itself does not specify how a name such as "agent://translator-zh-en" is bound to a particular peer or how that binding is disseminated across the network.

Without a name system, agents must exchange raw cryptographic peer identifiers — opaque strings that are neither memorable nor semantic. A name system provides the indirection layer that allows agent names to persist across peer-identity rotations, enables human operators to refer to agents by meaningful labels, and supports anycast routing where a single name resolves to the best available instance.

1.2. Scope

This document is one of four core Internet-Drafts in the Agent Network Protocol (ANP) suite: AIP [I-D.song-anp-aip] (datagram delivery), AITP [I-D.song-anp-aitp] (invocation transport), ANS (this document, name system), and ADP [I-D.song-anp-adp] (description and discovery). The four drafts are designed to co-evolve as a self-contained protocol suite; no additional specification is required for baseline interoperability. AIP's local-resolver semantic extension MAY be backed by an implementation that consults ADP discovery services for ranked capability matching; ANS core provides name-to-peer binding but does not itself perform ranked semantic discovery.

ANS is a name-layer protocol within the ANP suite. It sits between the human-facing name space and the peer-identity layer used by AIP.

This document specifies:

1. The agent:// URI syntax as used by ANS, refining the base grammar defined in AIP with namespace, version, and addressing-mode semantics. AIP remains the scheme authority ([I-D.song-anp-aip]); ANS defines name-layer interpretation only.

2. The Name Record format — a JSON object that binds an agent:// name to a peer identifier, skill tags, and temporal metadata.

3. Four AITP method names (ans.register, ans.resolve, ans.unregister, ans.lookup) for name operations over the Agent Invocation Transport Protocol (AITP) [I-D.song-anp-aitp].

4. A multi-layer resolution algorithm (local cache, persistent store, DHT, peer-assisted query).

5. Two dissemination mechanisms: GossipSub announcements for real-time propagation and DHT records for persistent storage.

6. The name lifecycle: registration, freshness, renewal, ownership, transfer, expiration, and release.

This document does not cover:

1. Agent capability description — specified by the Agent Description Protocol (ADP) [I-D.song-anp-adp].

2. Ranked capability discovery — the five-factor scoring algorithm is defined in ADP [I-D.song-anp-adp].

3. Economic mechanisms for name registration (token burn, auction, pricing) — these are deployment-specific policies.

4. DNS-based discovery (DNS-SD, well-known URIs) — discussed informatively in Appendix A.

5. Agent identity, key management, or credential issuance — these belong to AIP and deployment- context mechanisms.

1.3. Positioning

ANS is to agent:// URIs what DNS is to domain names: a name-to-address resolution service. ANS differs from DNS in three respects: (1) registrations are self-certified via Ed25519 signatures rather than delegated to registrars; (2) resolution is peer-to-peer (DHT + gossip) rather than hierarchical; (3) the name space supports anycast and channel addressing modes natively. Of these three modes, only unicast and anycast names are registered; channel names are derived from the URI syntax and map deterministically to GossipSub topics without creating a Name Record (see Section 7.4).

Unlike ADP's adp.discover, which ranks agents by a multi-factor scoring algorithm, ANS's ans.lookup performs tag-based filtering without scoring — a narrow-waist lookup analogous to DNS SRV queries. The two protocols are complementary: ANS provides name→peer binding; ADP provides capability→ranking.

1.4. Design Rationale

ANS is justified when the following conditions hold:

1. *Agents need stable, human-readable identifiers.*

   Cryptographic peer IDs (e.g., 12D3KooW...) are unsuitable for human reference, configuration files, and documentation. A name layer provides stability across peer-ID rotation.

2. *Multiple resolution paths improve availability.*

   A single DHT lookup may fail due to network partition. A layered resolution stack (local cache → persistent store → DHT → peer-assisted RPC) maximizes the probability of successful resolution.

3. *Anycast requires a name-to-set mapping.*

   When multiple instances serve the same logical service, the name system must resolve a single name to a set of candidates. Instance selection is then delegated to the caller or to a companion ranking protocol (ADP).

1.5. Assumptions

ANS assumes:

1. An AIP module providing agent:// datagram delivery with Ed25519-signed peer identities.

2. An AITP module capable of REQUEST/RESPONSE exchanges with the method names defined in this document.

3. A GossipSub implementation supporting topic-based pub/sub (e.g., libp2p GossipSub v1.1).

4. A Kademlia DHT implementation supporting namespace-prefixed key-value storage.

5. Name Record documents are UTF-8 encoded JSON ([RFC8259]).

1.6. Relationship to Adjacent Specifications

ANS occupies a narrow role in the ANP suite. This section clarifies the boundary between ANS and its sibling protocols to prevent misattribution of responsibilities.

Agent Internet Protocol (AIP) [I-D.song-anp-aip] AIP owns the agent:// URI scheme registration, the Ed25519 peer-identity model, and datagram delivery. ANS consumes the URI scheme and peer identities defined by AIP; it does not redefine them. The ABNF in Section 3.1 is a name-layer profile of the AIP base grammar, adding namespace, instance, and channel semantics for registration and resolution. The agent:// URI scheme registration belongs to AIP, not ANS.

Agent Invocation Transport Protocol (AITP) [I-D.song-anp-aitp] AITP provides the reliable REQUEST/RESPONSE transport over which ANS method calls are carried. AITP is a pure bearer: it provides segment framing, flow control, and status codes, but has no awareness of name semantics. ANS defines its own method names (ans.*) and error detail bodies; AITP carries them unchanged.

Agent Description Protocol (ADP) [I-D.song-anp-adp] ADP defines

Agent Cards (rich capability descriptions) and the adp.discover method (five-factor ranked discovery). ANS provides the narrower primitive: name-to-peer binding and tag-based lookup without scoring. The two protocols are complementary — ANS resolves names, ADP ranks capabilities — and SHOULD be deployed together.

Web-Native Discovery Profiles DNS-SD ([RFC6763]), .well-known URIs ([RFC8615]), and mDNS ([RFC6762]) are informative bridging mechanisms described in Appendix A. They are deployment profiles, not part of the ANS core protocol. An ANS implementation is fully conformant without supporting any web- native discovery profile.

1.7. Requirements Language

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

2. Terminology

Name Record A JSON object that binds an agent:// name to a peer identifier, skill tags, and temporal metadata. The Name Record is the unit of registration and resolution in ANS.

Peer ID The cryptographic identifier of a network node, derived from its Ed25519 public key. Used by AIP for datagram routing.

Namespace An optional organizational prefix in an agent:// URI that groups related agent names (e.g., "nlp" in "agent://nlp/translator").

Addressing Mode One of three resolution semantics for an agent:// URI: unicast (resolve to a specific instance), anycast (resolve to the best available instance of a service), or channel (map to a pub/sub topic for multicast delivery).

Owner The peer that first registered a Name Record. Ownership is immutable: only the Owner may update or transfer the record.

Sequence Number A strictly monotonically increasing integer in a Name Record, used to order updates and prevent replay of stale records.

3. agent:// URI Syntax

The agent:// URI scheme identifies agents in the ANP network. AIP [I-D.song-anp-aip] defines the base agent:// scheme and its core parsing rules. This section defines the ANS-specific naming interpretation and constraints used for registration and resolution. AIP [I-D.song-anp-aip] is the scheme authority and defines the base agent:// syntax and wire encoding; the grammar below is a compatible profile that adds namespace, instance, version, and addressing-mode semantics on top of the AIP base grammar.

3.1. ABNF Grammar

The following ABNF ([RFC5234]) defines the syntax of agent:// URIs. The grammar uses the core rules from [RFC5234] Appendix B and the generic URI components from [RFC3986].

```
agent-uri     = "agent://" agent-path [ "@" version ]
agent-path    = service-path / channel-path

; Service addressing (unicast or anycast)
service-path  = [ namespace "/" ] name [ "/" instance ]

; Channel addressing (trailing "/" required)
channel-path  = [ namespace "/" ] name "/"

namespace     = identifier
name          = identifier
instance      = identifier
version       = 1*( ALPHA / DIGIT / "." / "-" )

identifier    = lo-alpha-digit *( lo-alpha-digit / "-" )
lo-alpha-digit = %x61-7A / DIGIT   ; a-z / 0-9
```

Constraints:

* Each "identifier" segment MUST be 1 to 63 octets.

* The total agent-path (excluding "agent://") MUST NOT exceed 255 octets, matching the AIP MaxURILen.

* Identifiers MUST be lowercase ASCII letters, digits, and hyphens. Underscores are NOT RECOMMENDED; implementations MAY normalize underscores to hyphens on input.

* An identifier MUST NOT start or end with a hyphen.

3.2. Addressing Modes

An agent:// URI carries one of three addressing modes, determined by syntactic inspection of the path:

Mode	URI Pattern	Resolution Semantics
Unicast	agent://[ns/]name/instance	Resolve to exactly one peer: the named instance.
Anycast	agent://[ns/]name	Resolve to one or more peers registered under the service name; the caller or a companion protocol selects among candidates.
Channel	agent://[ns/]name/	Map to a GossipSub topic; datagrams are delivered to all subscribers. Channel names are derived, not registered — no Name Record is stored. See Section 7.4.

Table 1: Addressing Modes

Implementations MUST detect the addressing mode using the following algorithm:

```
if path ends with "/"           → Channel
else if path contains instance  → Unicast
else                            → Anycast
```

An agent:// URI with no namespace and no instance (e.g., "agent://translator-zh-en") is Anycast. This preserves backward compatibility with the flat names used by existing implementations.

3.3. Examples

```
# Anycast — route to any available translator
agent://translator-zh-en

# Anycast — namespace-scoped
agent://nlp/translator

# Unicast — specific instance
agent://nlp/translator/zh-en-01

# Unicast — versioned
agent://nlp/translator/zh-en-01@1.2.0

# Channel — multicast to all subscribers
agent://finance/market-updates/
```

3.4. Normalization

Implementations MUST normalize agent:// URIs before registration, resolution, and comparison by:

1. Converting the scheme and all identifier segments to lowercase.

2. Removing any trailing whitespace.

The normalized form retains the "agent://" prefix. Implementations that need a prefix-stripped key for DHT or database storage MAY derive it internally, but the canonical wire-format value is always the full URI.

Two URIs are considered equal if their normalized forms are byte-identical.

4. Name Record

A Name Record is a JSON object ([RFC8259]) that binds an agent:// name to a network peer and its metadata. Name Records are the unit of registration, dissemination, and resolution in ANS.

The following structural rules apply to every Name Record:

1. A Name Record MUST be a single JSON object ([RFC8259]) encoded in UTF-8.

2. A Name Record MUST NOT contain duplicate member names.

3. A Name Record MUST NOT contain members outside this specification unless they are placed inside the "extensions" object (see Section 4.4).

4. Implementations MUST ignore unknown top-level members for forward compatibility, but MUST preserve them when re-serializing the record (e.g., during gossip relay).

4.1. Record Fields

Field	Type	Req	Description
name	string	MUST	The complete agent:// URI, including the scheme prefix. E.g., "agent://nlp/translator" or "agent://translator-zh-en". Implementations that need a prefix-stripped key for storage or DHT lookups SHOULD derive it internally; the wire-format and object-model value is always the full URI.
peer_id	string	MUST	The registrant’s cryptographic peer identifier, as used by AIP for datagram routing.
namespace	string	MAY	The namespace segment extracted from the name (e.g., "nlp"). Redundant with "name" but enables efficient namespace-scoped queries.
skills	array	MAY	A JSON array of strings, each a lowercase skill tag (e.g., ["translation","nlp"]). Duplicate entries SHOULD be removed on input. Used for tag-based lookup (ans.lookup).
description	string	MAY	Free-text description of the agent’s purpose. Used for full-text search.
version	string	MAY	Semantic version string of the agent’s capability set (e.g., "1.2.0").
ttl	integer	MAY	Time-to-live in seconds. Default: 3600. Consumers SHOULD consider the record

			stale after this duration.
registered_at	string	MUST	Registration timestamp in RFC 3339 format ([RFC3339]).
expires_at	string	MUST	Expiration timestamp in RFC 3339 format. Records past this time MUST be treated as expired and SHOULD be removed from caches.
owner_id	string	MUST	The peer_id of the first registrant. Immutable after initial registration. Only the Owner may update or transfer the record.
seq	integer	MUST	Monotonically increasing sequence number. Receivers MUST reject records with a seq value less than or equal to the locally stored seq for the same name. See Section 4.3.
signature	string	MUST	Ed25519 signature ([RFC8032]) over the canonical record bytes. See Section 4.2. Encoded as unpadded Base64url ([RFC4648] §5).
extensions	object	MAY	A JSON object carrying deployment-specific extension data. See Section 4.4.

Table 2: Name Record Fields

4.2. Signature Computation

The signature field authenticates the Name Record and binds it to the registrant's Ed25519 key pair. The signature is computed as follows:

1. Construct the signing input by concatenating the following fields in order, each encoded as UTF-8 with a newline (0x0A) separator: name, peer_id, namespace, skills (JSON-serialized array, e.g., '["nlp","translation"]'; empty array '[]' if absent), description, version, ttl (decimal string), registered_at, expires_at, owner_id, seq (decimal string).

2. Compute the Ed25519 signature ([RFC8032]) over the signing input using the registrant’s private key.

3. Encode the 64-byte signature as unpadded Base64url ([RFC4648] §5).

Receivers MUST verify the signature before accepting a Name Record. Verification requires the Ed25519 public key corresponding to the peer_id. Implementations MUST reject records whose signature does not verify, whose peer_id does not correspond to the signing key, or whose owner_id does not match the peer_id (for initial registrations) or the stored owner_id (for updates).

4.3. Sequence Numbers

The seq field provides replay protection and consistent ordering of Name Record updates.

* The initial registration MUST set seq to 1.

* Each subsequent update MUST increment seq by at least 1.

* Receivers MUST reject a record if its seq is less than or equal to the seq of the currently stored record for the same name.

* Receivers MUST reject a record if the seq value is unreasonably large (implementation-defined threshold) to prevent seq-space exhaustion attacks.

4.4. Extension Mechanism

The "extensions" field provides a stable mounting point for deployment-specific data that does not belong in the core Name Record. Examples include anti-spam proof tokens, pricing metadata, transfer receipts, and deployment-policy hints.

* The value of "extensions" MUST be a JSON object.

* Each key within "extensions" SHOULD be a reverse-domain or URN-namespaced identifier to avoid collisions (e.g., "com.example.antispam").

* Implementations MUST preserve extension entries they do not understand when re-serializing or relaying a Name Record.

* Implementations MUST ignore extension entries they do not understand when processing a Name Record.

* The "extensions" object is excluded from the signature computation (Section 4.2). This is an intentional design trade-off: the core name-binding fields (name, peer_id, owner_id, seq, timestamps) are signed and tamper-evident — these fields fully cover the name-to-identity binding and temporal validity, so core record integrity does not depend on extensions. Extensions are left unsigned to allow deployment-local annotations and relay-added metadata (e.g., hop count, ingestion timestamp) that the originator cannot predict at signing time.

  Consequently, any peer on the relay path may add, modify, or remove extension entries. Deployment profiles that require stronger integrity for specific extensions SHOULD define an inner signature field within their own extension namespace.

4.5. Validation Rules

Implementations MUST validate Name Records on receipt, whether from ans.register requests, GossipSub messages, or DHT retrieval. The following rules apply:

ID	Rule	Level
VAL-01	"name" conforms to the agent:// ABNF (Section 3.1).	MUST
VAL-02	"peer_id" is non-empty and is a syntactically valid peer identifier.	MUST
VAL-03	"owner_id" is non-empty. For initial registrations, owner_id = peer_id. For updates, owner_id matches the stored value.	MUST
VAL-04	"expires_at" is strictly after "registered_at", and "expires_at" is in the future at time of receipt.	MUST
VAL-05	"ttl" is a positive integer (> 0). Default 3600 if absent.	MUST
VAL-06	"seq" is an integer >= 1. For updates, seq is strictly greater than the stored seq.	MUST
VAL-07	"skills" entries are unique, lowercase strings. Duplicates are removed silently.	SHOULD
VAL-08	"namespace", if present, matches the namespace segment extracted from "name".	MUST
VAL-09	"signature" verifies per Section 4.2.	MUST
VAL-10	The addressing mode implied by "name" is consistent with the operation context. Channel names (trailing "/") are derived, not registered, and MUST NOT be submitted to ans.register (see Section 7.4).	MUST

Table 3: Name Record Validation Rules

Records that fail any MUST-level rule MUST be rejected. Records that fail a SHOULD-level rule SHOULD be accepted after corrective normalization (e.g., deduplicating skills).

5. Protocol Operations

ANS defines four AITP method names for name operations. Each method uses an AITP REQUEST/RESPONSE exchange; the request and response bodies are JSON objects.

5.1. ans.register — Register a Name

An agent sends a REQUEST with Method = "ans.register" to bind an agent:// name to its peer identity. Only Unicast and Anycast names may be registered; Channel names are derived from the URI syntax and MUST NOT be submitted to ans.register (see Section 7.4).

Request body A Name Record as defined in Section 4. All MUST fields MUST be present.

Response body (Status = OK) A JSON object with the following fields:

* "registered": boolean — true if the record was accepted.
* "name": string — the normalized name.
* "seq": integer — the accepted sequence number.
* "expires_at": string — the accepted expiration timestamp.

Error responses

* INVALID_REQUEST (6) — malformed record, invalid signature, or constraint violation (e.g., name syntax, field length).
* UNAUTHORIZED (5) — the sender is not the owner of an existing record with the same name.
* BUSY (8) — the receiver has reached its capacity for stored Name Records.

The receiver MUST perform the following validation before accepting a registration:

1. Verify the agent:// name conforms to the ABNF in Section 3.1.
2. Verify the Ed25519 signature per Section 4.2.
3. If a record with the same name already exists, verify that the request's owner_id matches the stored owner_id and that the request's seq is strictly greater than the stored seq.
4. If no record exists, set the owner_id to the sender's peer_id.

5. Verify that expires_at is in the future and that ttl is positive.

Registration MAY additionally require a deployment-specific anti-spam mechanism such as proof-of-work, token expenditure, or rate limiting. Such mechanisms are outside the scope of this specification.

5.2. ans.resolve — Resolve a Name

A caller sends a REQUEST with Method = "ans.resolve" to look up the Name Record(s) bound to an agent:// name.

Request body A JSON object with the following fields:

* "name": string (MUST) — the full agent:// URI to resolve.

Response body (Status = OK) A JSON object with a uniform envelope containing three fields:

* "mode": string (MUST) — one of "unicast", "anycast", or "channel".

* "records": array (MUST) — matching Name Records. For Unicast: at most one element. For Anycast: zero or more elements, ordered by seq descending. For Channel: empty array.

* "topic": string or null (MUST) — the GossipSub topic string for Channel mode; null for Unicast and Anycast.

* "source": string (MAY) — when present, indicates the resolution layer that produced the result. Values are drawn from: "local" (in-memory cache), "store" (persistent database), "dht" (Kademlia DHT), "peer" (peer-assisted RPC). Omitted if unknown.

Error responses INVALID_REQUEST (6) if the name is malformed.

If the receiver does not have the requested record in its local store, it MAY attempt resolution via the DHT or peer-assisted query before returning an empty records array.

5.3. ans.unregister — Remove a Name

An agent sends a REQUEST with Method = "ans.unregister" to remove its Name Record.

Request body A JSON object with the following fields:

* "name": string (MUST) — the agent:// name to unregister.

* "signature": string (MUST) — Ed25519 signature over the UTF-8 string "unregister:" concatenated with the normalized name.

Response body (Status = OK) A JSON object: { "unregistered": true }.

Error responses
* UNAUTHORIZED (5) — the sender is not the owner.

* INVALID_REQUEST (6) — invalid signature or name not found.

The receiver MUST verify that the signature is valid and that the sender's peer_id matches the stored owner_id before removing the record. Upon successful unregistration, the receiver SHOULD propagate the removal via GossipSub (see Section 8.1).

5.4. ans.lookup — Tag-Based Name Lookup

A caller sends a REQUEST with Method = "ans.lookup" to find agents by skill tags. Unlike ADP's adp.discover, ans.lookup is a narrow-waist filter: it returns all matching records without scoring or ranking.

Request body A JSON object with the following fields:

Field	Type	Description
tags	string[]	Skill tags to match (at least one SHOULD be provided).
namespace	string	Optional namespace filter.
limit	integer	Maximum results. Default: 10.

Table 4: ans.lookup Request Fields

Response body (Status = OK) A JSON object with a "results" array. Each element contains:

* "record": the matching Name Record.

* "matched_tags": string[] — tags that contributed to the match.

Error responses INVALID_REQUEST (6) if the query is malformed.

A Name Record matches if at least one of the query tags appears in the record's skills array (after normalization). Implementations SHOULD normalize tags to lowercase and apply alias resolution (e.g., "py" → "python") before matching.

When the total number of matching records exceeds the AIP maximum message size (65535 octets), implementations SHOULD truncate the result set to "limit" entries and return only complete records that fit within a single response datagram. Callers that need additional results MAY issue subsequent queries with an "offset" parameter (application-level pagination) or use an AITP STREAM exchange for large result sets, as discussed in AITP [I-D.song-anp-aitp].

5.5. Method Summary

Method	Direction	Purpose
ans.register	Agent → Peer/Directory	Bind a name to a peer identity
ans.resolve	Caller → Peer/ Directory	Look up name → peer binding
ans.unregister	Agent → Peer/Directory	Remove a name binding
ans.lookup	Caller → Peer/ Directory	Tag-based name filter (no ranking)

Table 5: ANS AITP Methods

6. Error Handling

ANS methods reuse AITP status codes for transport-level outcome (e.g., OK, INVALID_REQUEST, UNAUTHORIZED). In addition, ANS defines a structured error detail body that provides protocol-specific reason codes.

6.1. Error Detail Envelope

When an ANS method returns an AITP error status, the response body SHOULD include the following JSON object:

```
{
  "code":   "ANS-1002",
  "title":  "invalid-signature",
  "detail": "Ed25519 signature verification failed for
             name 'agent://nlp/translator'.",
  "name":   "agent://nlp/translator"
}
```

Fields:

code An ANS error code from Table 6. MUST be present.

title A short, stable, machine-readable slug. MUST be present.

detail A human-readable explanation. MAY be present.

name The agent:// name that triggered the error, if applicable. MAY be present.

6.2. ANS Error Codes

Code	Title	AITP Status	Meaning
ANS-1001	invalid-name	INVALID_REQUEST	Name does not conform to agent:// ABNF.
ANS-1002	invalid-signature	INVALID_REQUEST	Ed25519 signature verification failed.
ANS-1003	owner-mismatch	UNAUTHORIZED	Sender's peer_id does not match the stored owner_id.
ANS-1004	stale-seq	INVALID_REQUEST	Record seq is less than or equal to the stored seq.
ANS-1005	expired-record	INVALID_REQUEST	Record's expires_at is in the past.

ANS-1006	malformed-record	INVALID_REQUEST	Record violates structural or validation rules (Section 4.5).
ANS-1007	unsupported-mode	INVALID_REQUEST	Addressing mode not supported by this operation (e.g., channel name in ans.register).
ANS-1008	capacity-exceeded	BUSY	Receiver has reached its storage capacity for Name Records.
ANS-1009	not-found	INVALID_REQUEST	No record exists for the requested name.

Table 6: ANS Error Code Registry

AITP status codes provide the transport-level disposition (success, client error, server error). ANS error codes provide the protocol-specific reason. A receiver MUST set the AITP status code; it SHOULD include the ANS error detail body for non-OK responses.

7. Name Resolution

Name resolution is the process of mapping an agent:// URI to one or more Name Records. ANS defines a multi-layer resolution algorithm that balances latency, consistency, and availability.

7.1. Resolution Algorithm

When resolving an agent:// URI, implementations SHOULD attempt the following layers in order, returning the first successful result:

1. *Layer 1 — Local Memory Cache.* Look up the normalized name in the in-memory Name Table (e.g., sync.Map). If a non-expired record is found, return it.

2. *Layer 2 — Persistent Store.* Query the local persistent store (e.g., SQLite) for a non-expired record. If found, populate the L1 cache and return.

3. *Layer 3 — DHT.* Query the DHT at key "/clawnet-ans/{normalized-name}". If a valid, non-expired record is found, store it in L2 and L1 and return.

4. *Layer 4 — Peer-Assisted Query.* Send an ans.resolve request to connected peers in parallel. Accept the first valid response. Store in L2 and L1.

If all layers fail, the resolver returns a not-found result.

Implementations MAY skip layers or query multiple layers in parallel to reduce latency. The layer order above is RECOMMENDED for the common case.

7.2. Unicast Resolution

A Unicast URI (agent://[ns/]name/instance) resolves to exactly one Name Record by exact-matching the full path "{namespace}/{name}/{instance}" or "{name}/{instance}". If no exact match is found, the resolver returns not-found.

7.3. Anycast Resolution

An Anycast URI (agent://[ns/]name) resolves to all non-expired Name Records whose name field matches the service name, either by exact match or by prefix match (all records sharing the same namespace and service segments).

The resolver returns the full candidate set. Instance selection is the caller’s responsibility; callers MAY use ADP’s ranked discovery (adp.discover) or any other selection strategy.

7.4. Channel Resolution

Channel names are derived, not registered. A Channel URI (agent://[ns/]name/) does not bind to a Peer ID and has no corresponding Name Record. Instead, the resolver deterministically maps the channel name to a GossipSub topic string using the following rule:

```
topic = "/clawnet/channel/" + normalized-channel-path
        (strip trailing "/")
```

Example:
agent://finance/market-updates/
→ /clawnet/channel/finance/market-updates

The resolver returns the topic string. The AIP module delivers datagrams to this topic via GossipSub pub/sub.

7.5. Cache Behavior

Resolved Name Records SHOULD be cached at both the memory layer (L1) and persistent layer (L2). The cache entry MUST expire no later than the Name Record's expires_at timestamp. Implementations SHOULD also re-resolve after the TTL period to obtain fresher records.

When a GossipSub announcement updates or removes a cached name (see Section 8.1), implementations MUST update or invalidate the L1 and L2 cache entries accordingly.

8. Dissemination

ANS provides two complementary dissemination mechanisms for Name Records: GossipSub for real-time propagation and DHT for persistent, query-driven retrieval.

8.1. GossipSub Announcements

Name registrations and unregistrations are announced via the GossipSub topic "/clawnet/ans".

8.1.1. Message Format

Each GossipSub message on "/clawnet/ans" is a UTF-8 JSON object with the following fields:

Field	Type	Description
action	string	"register" or "unregister".
record	object	The Name Record (for "register") or a partial record containing at least "name", "owner_id", and "signature" (for "unregister").

Table 7: GossipSub ANS Message Fields

8.1.2. Receiver Validation

Upon receiving a GossipSub ANS message, the receiver MUST:

1. Verify the signature on the Name Record per Section 4.2.

2. For "register": verify that seq is strictly greater than the locally stored seq for the same name (or that no local record exists). If valid, store or update the local cache and persistent store.

3. For "unregister": verify that the signature is valid and that the owner_id matches the stored record. If valid, remove the record from the local cache and persistent store.

4. Silently drop messages that fail verification. Implementations MUST NOT propagate invalid messages.

8.2. DHT Storage

Name Records are stored in the Kademlia DHT under the namespace "clawnet-ans". The DHT key for a Name Record is:

key = "/clawnet-ans/" + normalized-name

The DHT value is the UTF-8 JSON serialization of the Name Record. DHT put and get operations MUST validate the signature and sequence number using the same rules as GossipSub validation (Section 8.1.2).

DHT implementations SHOULD set a record expiration aligned with the Name Record's expires_at field. When a DHT get returns an expired record, the resolver MUST treat it as not found.

8.2.1. Related DHT Namespaces

The ANP suite uses four DHT namespaces. Only the ANS namespace is specified by this document; the others are documented here for cross-reference:

Namespace	Key Format	Purpose
/clawnet-ans/	/clawnet-ans/{name}	Name Records (this document)
/clawnet-profile/	/clawnet-profile/{peer_id}	Agent Cards (ADP)
/clawnet-rep/	/clawnet-rep/{peer_id}	Reputation records
/clawnet-txn/	/clawnet-txn/{txn_id}	Transaction records

Table 8: DHT Namespaces

8.3. Peer-Assisted Resolution

When Layers 1-3 of the resolution algorithm fail, the resolver MAY send ans.resolve requests to currently connected peers in parallel. This is Layer 4 of the resolution stack.

Implementations SHOULD apply the following constraints to peer-assisted resolution:

* A timeout of 5 seconds per peer query.

* Accept the first valid response (first-wins strategy).

* Limit fan-out to at most 5 concurrent peer queries to avoid network amplification.

* Validate the returned Name Record using the same signature and seq checks as other layers.

9. Name Lifecycle

9.1. Registration

A name is registered by sending an ans.register request to one or more peers or directory agents. Upon successful registration, the registrant SHOULD announce the Name Record via GossipSub and publish it to the DHT.

The initial registration sets owner_id = peer_id and seq = 1. This binding is immutable: subsequent updates MUST come from the same owner_id.

9.2. Freshness and TTL

The TTL field indicates how long a consumer should consider the record fresh after retrieval. The expires_at field provides the absolute expiration boundary.

Registrants SHOULD re-register (with incremented seq) before expires_at to maintain continuous name availability. Periodic re-broadcast via GossipSub (e.g., every 5 minutes) helps ensure propagation to newly joined peers.

9.3. Renewal

Renewal is a re-registration with the same name and owner_id, an incremented seq, and an updated expires_at. The ans.register method serves both initial registration and renewal — no separate method is required.

Deployment profiles MAY impose additional requirements for renewal (e.g., periodic payment, proof-of-work, activity checks). Such requirements are outside the scope of this specification.

9.4. Ownership and Transfer

Name ownership is established at first registration and is immutable. Only the Owner (identified by owner_id) may update or unregister a name.

Because owner_id is immutable while peer_id is mutable, an Owner MAY update a Name Record to point to a different serving peer_id (e.g., after key rotation or migration to a new host). The update MUST be signed by the key corresponding to the current owner_id and carry a strictly higher seq value. This mechanism allows names to persist across peer-identity rotations without re-registration.

ANS does not define a transfer protocol at the AITP layer. Deployment profiles MAY define transfer mechanisms (e.g., dual-signed transfer messages) as extensions. A transferred name results in a new registration with the new owner’s peer_id, owner_id, and seq reset to 1.

9.5. Expiration and Release

A Name Record is expired when the current time is past its expires_at timestamp. Expired records:

* MUST NOT be returned by ans.resolve or ans.lookup.
* SHOULD be removed from persistent stores during periodic garbage collection.
* SHOULD be removed from the DHT (or allowed to expire via DHT TTL).

Once expired and removed, the name returns to the available pool and may be registered by any agent. Deployment profiles MAY impose a grace period between expiration and release.

10. Security Considerations

10.1. Name Spoofing

An adversary may attempt to register a Name Record for a name owned by another agent. The owner_id immutability rule prevents this: once a name has an owner, only that owner's signed updates are accepted. Receivers MUST verify signatures and owner_id matching before accepting any registration or update.

10.2. Replay Attacks

An adversary may replay an old Name Record to revert the name to stale state (e.g., a peer_id that the owner has since rotated). The monotonic seq field prevents this: receivers MUST reject records with seq less than or equal to the locally stored value.

To limit seq-space exhaustion, implementations SHOULD reject seq values that exceed the locally stored seq by more than a deployment-defined threshold (e.g., 1000).

10.3. Name Squatting

An adversary may register many names to deny them to legitimate agents. This document does not prescribe a specific mitigation; deployment profiles SHOULD employ at least one of:

* Proof-of-work: require computational effort before accepting ans.register.
* Token expenditure: require payment (e.g., Shell burn) proportional to name length or scarcity.

* Rate limiting: limit the number of registrations per peer per time window.

* Activity requirements: expire names whose registrant has not been active for a deployment-defined period.

10.4. GossipSub Flooding

An adversary may flood the "/clawnet/ans" topic with invalid or high-rate messages. Implementations MUST validate every message before processing (see Section 8.1.2) and SHOULD apply per-peer message rate limiting. GossipSub mesh scoring and peer scoring (as defined in GossipSub v1.1) provide additional protection against flooding.

10.5. DHT Poisoning

An adversary may attempt to store malicious Name Records in the DHT. Implementations MUST verify signatures on DHT get results before accepting them. Implementations SHOULD verify that the record's seq is consistent with locally known state (if any) and SHOULD cross-validate DHT results against GossipSub-disseminated records when possible.

10.6. Name Enumeration

The ans.lookup method allows querying by tags, which may reveal the set of registered agents. Implementations that are concerned about enumeration MAY restrict ans.lookup to authenticated peers or limit result sizes.

10.7. Privacy

Name Records contain peer_id, skill tags, and description text, which reveal agent identity and capabilities. Agents that require privacy SHOULD use minimal skill tags and description text. Agents that require anonymity SHOULD NOT register names and should instead use direct peer-ID addressing.

11. IANA Considerations

11.1. URI Scheme

ANS relies on the "agent" URI scheme defined by AIP [I-D.song-anp-aip]. This document makes no independent URI scheme registration request; the agent:// scheme registration is the responsibility of the AIP specification.

11.2. ANS Error Codes

The ANS error codes defined in Table 6 are protocol- internal identifiers within the ANP suite and are not drawn from IANA-managed registries. Should a formal ANS error code registry be established by a future document, the codes in Table 6 are candidates for registration.

11.3. AITP Method Names

The AITP method names (ans.register, ans.resolve, ans.unregister, ans.lookup) are conventions within the ANP protocol suite and are not drawn from IANA-managed registries. Should a formal AITP method registry be established by a future document, these method names are candidates for registration.

12. Implementation Status

Per [RFC7942], this section records known implementations.

12.1. ClawNet

Organization ChatChatTech

URL https://github.com/ChatChatTech/ClawNet

Description Go implementation of the ANP suite. The name system module uses the historical internal name "LNS" in source code; this predates the ANS terminology but covers the same protocol scope. All protocol-facing identifiers use agent://.

Maturity Alpha

Coverage Spec sections implemented:

* Name Record (§4): all 12 core fields, Ed25519 signature, monotonic seq, immutable owner_id
* Resolution (§7): 4-layer stack (memory, SQLite, DHT, peer-assisted RPC with first-wins strategy)
* Dissemination (§8): GossipSub register/unregister with signature validation; DHT namespace with Ed25519 validation
* Lifecycle (§9): registration, renewal, expiration, proof-of-work anti-spam

* Lookup (§5.4): tag-based narrow-waist filter with alias normalization and standard tag vocabulary

* Full-text search via SQLite FTS5 (BM25) over name, namespace, skills, description

Under integration:

* AITP method binding (§5) — currently REST-only; AITP segment framing in progress

* Wire-level topic and DHT namespace migration from /clawnet/lns to /clawnet/ans

* Channel addressing mode (§7.4) + derived topic mapping

* Uniform resolve envelope (§5.2) — currently returns single best match for anycast

* extensions field (§4.5) — not yet surfaced in record struct

* Validation rule table (§4.6) — rules enforced ad hoc; consolidated validation pass pending

Language Go

License Open source

Contact ink@chatchat.space

13. References

13.1. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <https://www.rfc-editor.org/info/rfc2119>.

[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <https://www.rfc-editor.org/info/rfc8174>.

[RFC8259] Bray, T., "The JavaScript Object Notation (JSON) Data Interchange Format", STD 90, RFC 8259, DOI 10.17487/RFC8259, December 2017, <https://www.rfc-editor.org/info/rfc8259>.

[RFC8032] Josefsson, S. and I. Liusvaara, "Edwards-Curve Digital Signature Algorithm (EdDSA)", RFC 8032, DOI 10.17487/RFC8032, January 2017, <https://www.rfc-editor.org/info/rfc8032>.

[RFC4648] Josefsson, S., "The Base16, Base32, and Base64 Data Encodings", RFC 4648, DOI 10.17487/RFC4648, October 2006, <https://www.rfc-editor.org/info/rfc4648>.

[RFC3339] Klyne, G. and C. Newman, "Date and Time on the Internet: Timestamps", RFC 3339, DOI 10.17487/RFC3339, July 2002, <https://www.rfc-editor.org/info/rfc3339>.

[RFC3986] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", STD 66, RFC 3986, DOI 10.17487/RFC3986, January 2005, <https://www.rfc-editor.org/info/rfc3986>.

[RFC5234] Crocker, D., Ed. and P. Overell, "Augmented BNF for Syntax Specifications: ABNF", STD 68, RFC 5234, DOI 10.17487/RFC5234, January 2008, <https://www.rfc-editor.org/info/rfc5234>.

[I-D.song-anp-aip] Song, J., "Agent Internet Protocol (AIP)", Work in Progress, Internet-Draft, draft-song-anp-aip-01, March 2026, <https://datatracker.ietf.org/doc/html/draft-song-anp-aip-01>.

[I-D.song-anp-aitp] Song, J., "Agent Invocation Transport Protocol (AITP)", Work in Progress, Internet-Draft, draft-song-anp-aitp-00, March 2026, <https://datatracker.ietf.org/doc/html/draft-song-anp-aitp-00>.

13.2. Informative References

[RFC7942] Sheffer, Y. and A. Farrel, "Improving Awareness of Running Code: The Implementation Status Section", BCP 205, RFC 7942, DOI 10.17487/RFC7942, July 2016, <https://www.rfc-editor.org/info/rfc7942>.

[RFC6762] Cheshire, S. and M. Krochmal, "Multicast DNS", RFC 6762, DOI 10.17487/RFC6762, February 2013, <https://www.rfc-editor.org/info/rfc6762>.

[RFC6763] Cheshire, S. and M. Krochmal, "DNS-Based Service Discovery", RFC 6763, DOI 10.17487/RFC6763, February 2013, <https://www.rfc-editor.org/info/rfc6763>.

[RFC8615] Nottingham, M., "Well-Known Uniform Resource Identifiers (URIs)", RFC 8615, DOI 10.17487/RFC8615, May 2019, <https://www.rfc-editor.org/info/rfc8615>.

[I-D.song-anp-adp] Song, J., "Agent Description Protocol (ADP)", Work in Progress, Internet-Draft, draft-song-anp-adp-00, March 2026, <https://datatracker.ietf.org/doc/html/draft-song-anp-adp-00>.

Appendix A. Web-Native Discovery Profile (Informative)

This appendix describes how the agent:// name space can be bridged to web-native discovery mechanisms. These mechanisms are informative deployment profiles, not protocol requirements.

A.1. DNS-SD (RFC 6763)

An agent with a domain name may publish DNS SRV and TXT records for DNS-based Service Discovery ([RFC6763]):

```
; SRV record (RFC 2782)
_clawnet._tcp.acme.com. IN SRV 0 0 4001 agent1.acme.com.

; TXT record (RFC 6763 §6)
_clawnet._tcp.acme.com. IN TXT "peer=12D3KooW..."
                               "agent=agent://acme/main"
                               "skills=translation,nlp"
```

Resolvers that support DNS-SD may query these records to bootstrap ANP connectivity with enterprise agents.

A.2. Well-Known URI (RFC 8615)

An agent with an HTTPS domain may publish its Agent Card at a well-known URI:

```
GET https://acme.com/.well-known/agent-card.json
→ 200 OK
→ Content-Type: application/json
→ { Agent Card JSON (ADP) }
```

This enables web-based clients to discover ANP agents without participating in the P2P network.

A.3. Local Discovery (mDNS)

For LAN discovery, agents may publish mDNS ([RFC6762]) service records:

```
_clawnet._tcp.local. IN SRV 0 0 4001 agent1.local.
_clawnet._tcp.local. IN TXT "peer=12D3KooW..."
                            "skills=translation,nlp"
```

Appendix B. Implementation Alignment Notes

This appendix documents the relationship between this specification and the reference implementation in the ClawNet codebase. Go source symbols use the legacy prefix "LNS" (from the pre-ANS development phase); these are internal identifiers, not protocol-level names.

Spec Concept	Go Symbol / Location	Notes
Name Record	lob.LNSRecord (lob-protocol/lns.go)	12 fields mapping to ANS Name Record
GossipSub Message	lob.LNSGossipMessage	action + record fields
ANS Store Interface	lob.LNSStore (5 methods)	RegisterLNS, ResolveLNS, DeleteLNS, SearchLNS, ListLNS
SQLite Store	store.LNS* methods (store/lns.go)	lns_records table + lns_fts FTS5
agent:// URI Parser	aip.ParseURI (aip/types.go)	2-segment: namespace/name@version
Discovery Algorithm	lob.Discover (lob-protocol/discovery.go)	5-factor scoring (ADP §6 alignment)
Narrow-Waist Lookup	lob.Lookup (lob-protocol/discovery.go)	Tag-match only, no scoring
DHT Namespace	/clawnet-ans/{path}	Kademlia DHT with Ed25519 validation; {path} is the prefix-stripped key derived from the full URI. Implementation

		currently uses legacy /clawnet-lns/ prefix
GossipSub Topic	/clawnet/ans	Register/unregister announcements
REST API	daemon.handleLNS* (daemon/lns_rpc_api.go)	14 endpoints on localhost:3998
CLI	cli.LNS* (cli/lns.go)	register, resolve, list, discover, alias
Standard Tags	discovery.StandardTags (60+ tags)	20+ alias mappings
Proof-of-Work	daemon.handleLNSRegister	SHA256, 20 leading zero bits
Peer-Assisted Query	daemon.handleLNSResolve (Layer 4)	Parallel RPC, 5s timeout, first-wins

Table 9: Spec-Implementation Mapping

Authors' Addresses

Jinke Song
Dept. of CSE, Hong Kong University of Science and Technology
Email: ink@chatchat.space

Mu Yuan
Dept. of IE, The Chinese University of Hong Kong
Email: muyuan@cuhk.edu.hk