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

J. Song
HKUST
M. Yuan
CUHK

25 March 2026

Agent Internet Protocol (AIP)
draft-song-anp-aip-00

Abstract

The Internet Protocol (IP, RFC 791) provides best-effort datagram delivery between hosts identified by numeric addresses. The Agent Internet Protocol (AIP) provides best-effort datagram delivery between autonomous AI agents identified by agent:// URIs -- human-readable, capability-descriptive names.

AIP is designed to serve as the narrow waist of the Agent Network Protocol (ANP) suite, occupying an architectural position analogous to IP in the TCP/IP suite: a minimal common substrate through which upper-layer protocols and link technologies interoperate without direct coupling.

This document specifies the AIP message format, addressing model, upper-layer protocol demultiplexing, TLV options, message processing rules, error handling, and interfaces to adjacent protocol layers. Name registration, distributed name resolution, and semantic discovery are provided by a companion resolver service and are outside the scope of this document.

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

The Internet Protocol [RFC0791] provides a universal mechanism for delivering datagrams between hosts identified by fixed-length numeric addresses. IP was designed under the assumption that communication endpoints are hardware devices with stable locations on physical networks.

In agent networking, the endpoints are not hardware devices but software agents: autonomous programs that can be created, destroyed, migrated, and replicated at will. They do not have fixed locations. They are identified not by physical attachment points but by their names and capabilities.

An AI agent that needs a French-to-Japanese translator does not know -- and should not need to know -- any IP address. It needs to reach "agent://translation/fr-ja". In AIP, names serve as addresses: rather than resolving a name to a numeric location before sending, the agent addresses messages directly to a human-readable URI and relies on a local resolver to map that name to a reachable peer.

Current naming systems such as the Domain Name System [RFC1035] map human-readable names to network locations. Agents need a naming system that maps intents to capabilities. AIP provides the datagram substrate: name-based addressing with the agent:// URI scheme and best-effort message delivery. The name resolution service itself -- including registration, distributed lookup, and semantic search -- is provided by a companion resolver that runs above AIP as an upper-layer protocol (one such resolver is ANS [I-D.song-anp-ans]).

1.2. Scope

This document is one of four core Internet-Drafts in the Agent Network Protocol (ANP) suite: AIP (this document, datagram delivery), AITP [I-D.song-anp-aitp] (invocation transport), ANS [I-D.song-anp-ans] (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 (Section 6.2) 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.

AIP defines an experimental datagram protocol serving as the narrow waist of the ANP suite, analogous to IP in TCP/IP. It provides a best-effort datagram service between named agents.

AIP provides:

1. Name-based addressing using agent:// URIs.

2. Best-effort datagram delivery.

3. Upper-layer protocol demultiplexing via the Protocol field.

4. TTL-bounded relay with deduplication.

5. Extensible TLV options for timestamps, tracing, and priority.

6. Error reporting for undeliverable messages.

AIP does NOT provide:

1. Reliable delivery -- datagrams may be lost, duplicated, or reordered. Reliability is the responsibility of a higher-layer protocol such as AITP [I-D.song-anp-aitp].

2. Ordering or flow control -- these are higher-layer functions.

3. Name registration, distributed lookup, or semantic discovery -- AIP does not define these functions. They are expected to be provided by a companion naming service; ANS [I-D.song-anp-ans] is one such architecture.

4. Fragmentation -- messages exceeding the link MTU are rejected, not fragmented.

5. Payload confidentiality -- encryption is the responsibility of a higher-layer protocol or the application.

1.3. Assumptions

AIP assumes:

1. A local resolver service capable of mapping agent:// URI tokens to link-layer delivery targets. AIP does not require any specific resolver architecture.

2. A link layer that can deliver variable-length byte sequences between peers identified by peer ID.

3. AIP MAY be used with higher-layer protocols that provide reliability, ordering, or application semantics, but does not depend on them for baseline datagram operation.

1.4. 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

Agent An autonomous software entity that participates in the ANP network. An agent has a cryptographic identity (an Ed25519 key pair) and registers one or more agent:// URIs.

Name The identifier component of an agent:// URI that appears after the optional namespace and slash. For example, in "agent://acme/code-reviewer", the name is "code-reviewer".

Namespace An optional organizational prefix in an agent:// URI that groups related agents. For example, in "agent://acme/code-reviewer", the namespace is "acme". A URI without a namespace is in the global namespace.

Resolution The process of mapping an agent:// URI to a peer ID so that AIP can transmit a message. Resolution is performed by a local resolver.

Resolver A local service that maps agent:// URIs to peer IDs. AIP delegates all resolution to the resolver. A resolver MAY additionally support semantic queries (see Section 6.2).

AIP Message The protocol data unit of AIP. An independent, self-contained datagram carrying a fixed header, source and destination agent:// URIs, optional TLV options, and a payload.

Peer ID A globally unique identifier for a network participant, derived from its Ed25519 public key. Used at the link layer for point-to-point addressing.

Node A machine (physical or virtual) running one AIP module. A node has one peer ID. Multiple agents MAY run on a single node.

In summary: a node has one link-layer Peer ID; one or more agents MAY run on that node; AIP addresses agents by agent:// URI, not by Peer ID.

3. The agent:// URI Scheme

The agent:// URI is the fundamental addressing primitive of AIP. It identifies an agent by a human-readable, capability-descriptive name rather than by a numeric network address. The scheme is registered with IANA in accordance with [RFC7595] (see Section 10.1).

An agent:// URI is a *routing token* designating a capability endpoint; it does not necessarily correspond to a singular legal identity. Multiple agents, instances, or replicas MAY be reachable through the same URI via resolver-level load balancing or failover. Conversely, a single agent MAY register multiple URIs for different capabilities.

3.1. Syntax

The syntax of agent:// URIs is defined in ABNF [RFC5234], following the generic URI structure of [RFC3986]:

```
agent-uri    = "agent://" agent-path
agent-path   = [ namespace "/" ] name [ "@" version ]
namespace    = ident
name         = ident
ident        = LCALPHA-DIGIT *( LCALPHA-DIGIT / "-" )
version      = 1*( DIGIT / "." / "-" / ALPHA )
LCALPHA-DIGIT = %x61-7A / DIGIT    ; a-z / 0-9
```

Names and namespaces MUST begin with a lowercase letter or digit, MUST NOT end with a hyphen, and MUST be at least one character long. Uppercase letters are not permitted; implementations MUST reject URIs containing uppercase letters rather than silently normalizing them.

The total URI length (including the "agent://" prefix) MUST NOT exceed 263 octets (255 for the wire-encoded path plus 8 for the "agent://" prefix).

3.2. Comparison Rules

Because all characters in agent:// URIs are already constrained to lowercase, comparison is a simple octet-by-octet match of the wire-encoded forms after the following normalization:

1. Remove any trailing slash.

2. Remove any trailing "@" not followed by a version string.

No percent-decoding is performed because the agent:// scheme does not permit percent-encoded characters.

3.3. Wire Encoding

On the wire, agent:// URIs are encoded WITHOUT the "agent://" prefix. The prefix is implicit -- all addresses in AIP messages are agent:// URIs. The encoding is UTF-8.

The encoding function strips the 8-octet "agent://" prefix:

* wire_encode("agent://acme/translator") -> "acme/translator" (15 octets)

* wire_encode("agent://translator") -> "translator" (10 octets)

* wire_encode("agent://x/y@1.0") -> "x/y@1.0" (7 octets)

The decoding function prepends "agent://":

* wire_decode("acme/translator") -> "agent://acme/translator"

This avoids repeating the scheme prefix for both source and destination addresses on the wire.

3.4. Examples

Valid agent:// URIs:

* agent://translator -- global namespace, name only

* agent://acme/code-reviewer@2.1 -- namespace "acme", name "code-reviewer", version "2.1"

* agent://openai/gpt-assistant -- namespace "openai", name "gpt-assistant"

* agent://research/arxiv-search -- namespace "research", name "arxiv-search"

3.5. Design Rationale

A dedicated URI scheme is used rather than an existing scheme because agent:// URIs serve as first-class routing tokens, not as locators that resolve to network endpoints. Existing alternatives were considered and found insufficient:

* DID URIs (did:method:...) are identity-centric and lack a built-in namespace/version structure for capability addressing.

* HTTPS well-known paths (https://host/.well-known/agent/...) couple agent identity to DNS names and TLS certificates, reintroducing the topology-bound addressing that AIP avoids.

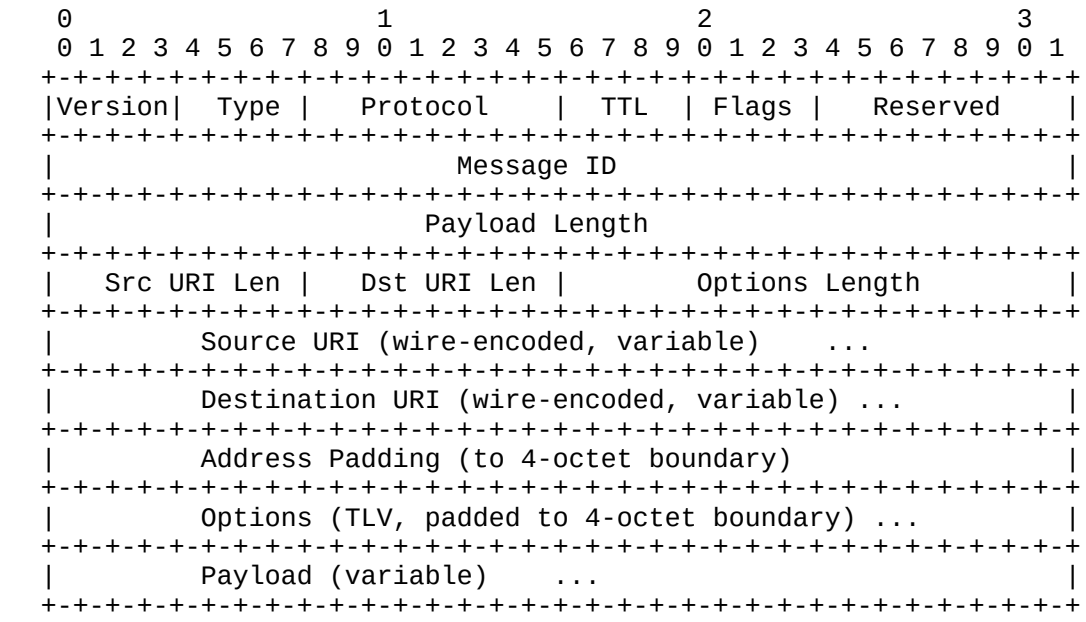
* Opaque identifiers with external resolvers provide no human-readable semantics and require mandatory resolver infrastructure before any message can be sent.

The agent:// scheme provides compact, human-readable routing tokens with a fixed wire encoding convention (Section 3.3).

4. AIP Message Format

AIP messages are carried as payloads of link-layer frames. An AIP message consists of a 16-octet fixed header, a variable-length address block containing the source and destination URIs (padded to a 4-octet boundary), a variable-length TLV options region, and a variable-length payload.

4.1. Header Structure



Note: one tick mark represents one bit position.

Figure 1: AIP Message Header (16 octets fixed)

4.2. Field Definitions

Version (4 bits) Protocol version. The current version is 1. Implementations MUST silently discard messages with an unrecognized version.

Type (4 bits) The message type:

Value	Name	Description
0	DATA	Normal data message (payload delivery to upper layer).
1	ERROR	Error report (carries an error payload, see Section 7.3).
2	PING	Liveness probe. Recipient MUST reply with PONG.
3	PONG	Liveness reply. Carries the Message ID of the original PING.
4-15		Reserved for future use.

Table 1: AIP Message Types

Implementations MUST silently discard messages with unrecognized Type values.

Protocol (8 bits) Identifies the upper-layer protocol that will process the payload. This field is analogous to the Protocol field in IPv4 [RFC0791]. It enables AIP to demultiplex incoming messages to the correct upper-layer handler.

Value	Name	Description
0	NONE	No upper-layer protocol (used for PING/PONG/control messages).
1	AITP	Agent Invocation Transport Protocol [I-D.song-anp-aitp].
2	ANS	Agent Name System [I-D.song-anp-ans].
3	ADP	Agent Description Protocol [I-D.song-anp-adp].
4-254		Available for future assignment (IANA registry).
255	EXPT	Experimental / private use.

Table 2: AIP Protocol Numbers

The Type and Protocol fields serve orthogonal purposes. Type identifies AIP's own control semantics (data delivery, error reporting, liveness probing). Protocol identifies which upper-layer consumer processes the payload. Thus, PING/PONG are AIP-layer operations (Type) that carry no upper-layer payload, while AITP, ANS, and ADP are upper- layer consumers (Protocol) whose payloads travel inside DATA messages (Type = 0).

In normal operation, ANS and ADP methods are carried as AITP segments (Protocol = 1, AITP). Protocol = 2 (ANS) and Protocol = 3 (ADP) are reserved for future use cases where a lightweight, single-datagram exchange bypasses AITP association overhead — for example, bootstrap resolution when no AITP association can yet be established, or one-shot name announcements in resource-constrained environments. Until such uses are formally specified, conforming implementations SHOULD use Protocol = 1 (AITP) for all ANS and ADP exchanges, and MUST silently discard messages with Protocol values 2 or 3 if they do not implement direct-datagram handlers for those protocols.

TTL (4 bits) Time to Live. An upper bound on the number of relay hops. Each relaying node MUST decrement the TTL by 1. When the TTL reaches zero, the message MUST be discarded; if the ERR flag is set, an ERROR message with code TTL_EXPIRED SHOULD be returned to the source.

The default TTL is 8. Valid range is 0-15. A TTL of 0 means the message MUST NOT be relayed; it is delivered only if the destination is local.

Flags (4 bits) A bitmask of control flags:

Bit	Value	Name	Description
3 (MSB)	0x8	SIG	Message carries an Ed25519 signature (64 octets) appended after the payload.
2	0x4	ERR	Error report requested. If delivery fails, an ERROR message SHOULD be returned to the source.
1	0x2	SEM	Semantic resolution provenance. When clear, AIP performs exact URI resolution (the baseline). When set, the message was delivered via resolver-assisted semantic discovery: AIP resolved a natural-language query through DISCOVER before transmission. The Destination URI field carries the resolved agent:// URI; the original query is recorded in the SemQuery option (Type 5, Section 4.3). Whether the resolver supports semantic resolution is deployment-specific (see Section 5.2).
0 (LSB)	0x1	RLY	Relay permitted. If set, intermediate nodes MAY forward this message toward the destination.

Table 3: AIP Flag Bits

Reserved (8 bits) Reserved for future use. Senders MUST set this field to zero. Receivers MUST ignore this field.

Message ID (32 bits) A sender-assigned identifier for deduplication and request-response correlation. Implementations MUST ensure uniqueness among outstanding messages to the same destination. A monotonically increasing counter is one possible strategy.

Payload Length (32 bits) The length in octets of the payload (not

including the header, address block, or options). Maximum value is 65535; implementations MUST reject messages with Payload Length exceeding 65535.

Src URI Len (8 bits) The length in octets of the wire-encoded Source URI. A value of 0 is permitted only in ERROR messages (system-generated error with no agent source). Maximum 255.

Dst URI Len (8 bits) The length in octets of the wire-encoded Destination URI. MUST NOT be 0. Maximum 255.

Options Length (16 bits) The total length in octets of the TLV options region, including any padding. If no options are present, this field is 0. Maximum 65535. The options region MUST be padded to a 4-octet boundary.

Source URI (variable, 0-255 octets) The agent:// URI of the sender in wire-encoded form (without the "agent://" prefix).

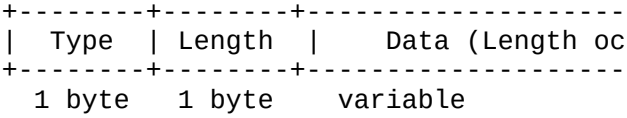
Destination URI (variable, 1-255 octets) The agent:// URI of the intended recipient in wire-encoded form (without the "agent://" prefix). This field always contains a valid wire-encoded agent:// URI, regardless of whether the SEM flag is set. When the SEM flag is set, AIP resolves the query before transmission and places the resolved URI here; the original query is carried in the SemQuery option (Section 4.3).

Address Padding (0-3 octets) Zero-valued padding octets added after the Destination URI to bring the combined address block (Src URI + Dst URI + padding) to a 4-octet boundary.

4.3. TLV Options

The options region carries zero or more Type-Length-Value (TLV) encoded options. Options provide extensibility without modifying the fixed header.

Each option is encoded as:



Exception: Type 0 (Pad1) is a single zero octet with no Length or Data fields; it is used for single-octet alignment padding.

Type	Name	Length	Description
0	Pad1	0	Single-octet padding (no Length field).
1	PadN	N	Multi-octet padding (N zero octets).
2	Timestamp	8	UTC Unix timestamp in microseconds (uint64, big-endian). Used for latency measurement and replay rejection.
3	Trace	variable	Opaque trace context blob for distributed tracing. Implementations MAY use W3C Trace Context traceparent format or any other tracing scheme.
4	Priority	1	Message priority hint (0 = lowest, 255 = highest). Receivers MAY use this for scheduling.
5	SemQuery	variable	UTF-8 semantic query string. MUST be present when the SEM flag is set; MUST NOT be present when the SEM flag is clear. Records the original natural-language query used for resolver-assisted discovery.
6-127			Available for future assignment (IANA).
128-254			Available for experimental / private use.
255			Reserved.

Table 4: Standard AIP Options

The entire options region MUST be padded to a 4-octet boundary using Pad1 or PadN options. Implementations MUST ignore unrecognized option types (skip by reading Type and Length, then advancing Length octets).

4.4. Message Signature

When the SIG flag (bit 3) is set, the message carries a 64-octet Ed25519 signature appended immediately after the payload. The signature MUST NOT be included in the Payload Length field; receivers detect its presence via the SIG flag.

The signature is computed over:

```
sign_input = fixed_header (16 octets, with Reserved set to 0)
           || src_uri_wire
           || dst_uri_wire
           || options_data (without padding)
           || payload
```

The signing key is the Ed25519 private key corresponding to the source agent's identity. Verification uses the public key bound to the source agent:// URI; the receiver obtains this key from its local resolver (see Section 9.2). The node's link-layer Peer ID is used only for message delivery, not for signature verification.

Implementations SHOULD set the SIG flag on all messages in production deployments. Implementations MUST verify signatures on incoming SIG-flagged messages and MUST discard messages with invalid signatures.

5. Local Resolver

AIP delegates all name resolution to a local resolver service. The resolver maps agent:// URIs to peer IDs that AIP can use for link-layer transmission.

AIP makes no assumptions about resolver internals. Any service that satisfies the Resolver interface (Section 5.1) is acceptable. ANS [I-D.song-anp-ans] is one companion resolver architecture.

5.1. Resolver Interface

The resolver MUST provide:

RESOLVE(uri) -> (peer_id, ok) Maps a fully-qualified agent:// URI to a peer ID. Returns ok = false if the URI cannot be resolved.

IS_LOCAL(uri) -> bool Returns true if the URI is registered by an

agent on the local node.

The resolver MAY use any local data structure (cache, table, or database) to satisfy these operations. AIP does not constrain resolver internals, storage model, propagation mechanism, or ranking method. Additional resolver operations such as registration and unregistration are outside AIP's scope and are defined by the resolver specification.

5.2. Optional Semantic Resolver Extension

A resolver MAY additionally support the following operation for use with the SEM flag (Section 6.2):

DISCOVER(query, tags, namespace, limit) -> list of (uri, peer_id, score) Maps a natural-language query to a ranked list of matching agents. A resolver that does not support semantic resolution MUST return an empty list.

Implementations that do not support semantic resolution are fully conformant with this specification.

5.3. Requirements on Implementations

Every AIP module MUST be initialized with a resolver. If no full resolver is available, a minimal implementation that only supports exact lookup of locally registered URIs is sufficient.

The resolver SHOULD cache results to avoid repeated lookups for the same URI. Cache entries SHOULD expire based on a TTL provided by the resolver. Implementations MUST enforce a maximum cache size to prevent unbounded memory growth; deployment-specific defaults are expected.

6. Name Resolution

AIP resolves destination URIs by calling the local resolver (Section 5.1). Baseline AIP operation uses exact destination tokens; resolver-assisted semantic discovery is an optional extension that occurs before forwarding. Once a destination URI has been selected, AIP forwarding proceeds identically in both cases.

The resolution strategy depends on the SEM flag:

6.1. Exact Resolution (SEM flag clear)

When the SEM flag is clear, the Destination URI is a fully- qualified agent:// URI. AIP calls RESOLVE(uri) on the local resolver. If the resolver returns ok = true, AIP uses the returned peer ID for transmission. If ok = false, AIP returns NAME_NOT_FOUND to the upper layer.

The resolver MAY use local or distributed mechanisms to satisfy the lookup; these mechanisms are outside AIP’s scope.

6.2. Semantic Resolution (SEM flag set) -- Optional Extension

Semantic resolution is an *optional extension*. AIP implementations are fully conformant without it. A resolver that does not support semantic resolution MUST return an empty list from DISCOVER, causing AIP to return NAME_NOT_FOUND.

When the SEM flag is set and the resolver supports semantic resolution, AIP calls DISCOVER(query, ...) on the local resolver and uses the highest-scoring result. AIP places the resolved agent:// URI in the Destination URI field and records the original query string in the SemQuery option (Type 5). If the resolver returns an empty list, AIP returns NAME_NOT_FOUND to the upper layer.

AIP does not specify any particular scoring algorithm. The scoring strategy -- including text similarity, tag matching, freshness, and trust signals -- is defined by the resolver implementation. Appendix A provides an informative example using one possible scoring formula.

A failed semantic query (empty result list) causes AIP to return NAME_NOT_FOUND. AIP does not perform query rewriting, fallback searches, or multi-step retrieval; such behavior, if desired, is the responsibility of the resolver or the application.

After resolution, AIP performs ordinary exact-URI delivery; semantic interpretation is confined to resolver selection and is not part of the forwarding semantics. The SEM flag does not alter forwarding behavior; it records resolver-assisted destination selection provenance for policy, debugging, and audit purposes.

7. Message Processing

This section defines AIP's complete processing model. The send and receive paths described below are self-contained: they depend only on the local resolver interface (Section 5.1) and the link-layer TRANSMIT/DELIVER primitives (Section 8.2). No knowledge of companion protocols is required to implement these paths.

7.1. Sending

When the upper layer calls SEND:

1. *Validate the source URI.*

   The source URI MUST be locally registered (IS_LOCAL returns true). If not, the send operation MUST fail.

2. *Resolve the destination.*

   If the SEM flag is clear, call RESOLVE(destination) on the local resolver. If the SEM flag is set, call DISCOVER(destination, ...) and use the highest-scoring result; place the resolved agent:// URI in the Destination URI field and record the original query in a SemQuery option (Type 5). If resolution fails, return NAME_NOT_FOUND to the upper layer.

3. *Construct the message.*

   Build the 16-octet fixed header with Version = 1, Type, Protocol, TTL (default 8), Flags, Reserved = 0, auto- generated Message ID, Payload Length. Wire-encode source and destination URIs. Encode any options (including the SemQuery option if SEM flag is set).

4. *Sign (if SIG flag set).*

   Compute the Ed25519 signature per Section 4.4 and append it after the payload.

5. *Transmit.*

   Serialize and call the link layer's TRANSMIT with the resolved peer ID and the serialized bytes.

7.2. Receiving

When the link layer delivers bytes:

1. *Parse and validate.*

Decode fixed header. If Version is unrecognized, discard. If Type is unrecognized, discard. If the total data is shorter than the computed message size, discard.

2. *Verify signature (if SIG flag set).*

   Extract the 64-byte signature. Verify against the public key bound to the source agent:// URI, as obtained from the local resolver (see Section 9.2). If verification fails, discard. If ERR flag is set, SHOULD send INVALID_SIGNATURE error.

3. *Deduplicate.*

   Check (Source URI, Message ID) against a bounded deduplication cache. If a match exists, discard. Otherwise, add to cache.

4. *Check TTL.*

   If TTL is 0 and destination is not local, discard. If ERR flag is set, send TTL_EXPIRED error.

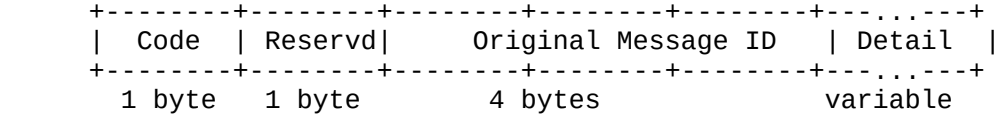
5. *Deliver or relay.*

   If IS_LOCAL(destination) returns true, decrement TTL and dispatch to the upper-layer handler identified by the Protocol field. If not local and RLY flag is set and TTL > 0, decrement TTL, call RESOLVE(destination) to obtain the next-hop peer, and re-transmit. If not local and RLY is not set, discard.

7.3. Error Handling

When an error occurs and the ERR flag is set, the AIP module SHOULD generate an ERROR message (Type = 1) to the original source. ERROR generation is best-effort; it is not guaranteed and MAY itself be suppressed under overload or resource constraints. ERROR messages MUST NOT be generated in response to other ERROR messages (preventing loops).

The error payload is encoded as:



Code	Name	Description
0		Reserved (MUST NOT be used).
1	NAME_NOT_FOUND	Destination URI could not be resolved.
2	TTL_EXPIRED	Message TTL reached zero before delivery.
3	MSG_TOO_LARGE	Message exceeds maximum size (65535 octets payload).
4	INVALID_SIGNATURE	Ed25519 signature verification failed.
5	RATE_LIMITED	Sender exceeded rate limit.
6	PROTOCOL_ERROR	Malformed message or protocol violation.
7	SHUTTING_DOWN	Receiving node is shutting down gracefully.
8	INTERNAL_ERROR	Unspecified internal processing error.
9-255		Available for future assignment (IANA).

Table 5: AIP Error Codes

The Original Message ID field MUST carry the full 32-bit Message ID of the message that triggered the error, enabling precise correlation. The Detail field is a UTF-8 string providing additional context; it MAY be empty.

7.4. Deduplication

In a GossipSub topology, messages may arrive via multiple paths. Implementations MUST maintain a deduplication cache keyed by (Source URI, Message ID). Messages matching a cache entry MUST be silently discarded. Implementations MUST bound the cache size and entry lifetime to prevent unbounded resource consumption; deployment-specific defaults are expected.

7.5. Rate Limiting

Implementations MUST enforce per-peer rate limiting to prevent denial-of-service. Specific thresholds are deployment-specific; implementations SHOULD document their default values. Messages exceeding the rate limit SHOULD be discarded; if the ERR flag is set, a RATE_LIMITED error SHOULD be returned.

8. Interfaces

8.1. Upper Layer Interface (AITP and others)

AIP provides the following service primitives to upper-layer protocols:

SEND(source, destination, protocol, payload, options) -> message_id Sends a datagram.

- *source* -- agent:// URI. MUST be locally registered.
- *destination* -- an agent:// URI token for baseline delivery; when the SEM flag is requested, the caller MAY provide a semantic query string, which the local resolver MUST resolve to an agent:// URI before transmission.
- *protocol* -- upper-layer protocol number (0-255).
- *payload* -- opaque bytes.
- *options* -- optional: type, ttl, flags, message_id, TLV options list.

Returns: Message ID, or error.

RECEIVE callback(source, destination, protocol, payload, message_id, options) Invoked when a message arrives for a locally registered agent. The Protocol field identifies which upper-layer handler receives the callback.

Note: Registration, unregistration, and semantic discovery are resolver operations, not AIP primitives.

8.2. Lower Layer Interface (Link Layer)

TRANSMIT(peer_id, raw_bytes) Sends bytes to a peer. The link layer routes them using whatever technology is available (libp2p, TCP, QUIC, WS, Bluetooth, etc.).

DELIVER callback(peer_id, raw_bytes) Invoked by the link layer for each received message. AIP registers this callback during initialization.

AIP makes no assumptions about the link layer beyond these two primitives.

9. Security Considerations

9.1. Message Authentication

When the SIG flag is set, AIP messages carry an Ed25519 signature providing origin authentication and integrity protection. Implementations SHOULD set the SIG flag on all messages. Implementations MUST verify signatures before processing signed messages and MUST discard messages with invalid signatures.

Signature verification is performed against the public key bound to the source agent:// URI; node Peer IDs are link- layer artifacts and are not used for application-layer signature verification. The receiver obtains the URI-to-key binding from its local resolver (see Section 9.2).

9.2. Identity and Signing Key Relationship

Each agent possesses its own Ed25519 key pair. A node’s link-layer Peer ID is derived from one Ed25519 public key, but multiple agents MAY run on a single node, each with a distinct key pair and agent:// URI.

When multiple agents share a node, the message signature binds the payload to the *agent’s* key, not the node’s Peer ID. The receiver obtains the agent’s public key from the resolver (which maintains the URI-to-public-key binding).

Relay nodes that forward messages on behalf of others need not verify the signature; only the final destination endpoint MUST verify. Relay nodes MAY verify opportunistically if they have access to the signing key.

9.3. Name Spoofing

AIP relies on the resolver to verify that agent:// URIs are bound to legitimate Ed25519 public keys. The specifics of name registration verification are defined by the resolver specification.

9.4. Resolution Poisoning

A malicious resolver could return forged peer IDs, directing messages to an attacker. Mitigations:

* Implementations SHOULD use the SIG flag so that receivers can verify message authenticity independent of resolution correctness.

* Resolver implementations SHOULD require Ed25519 signatures on all name entries.

9.5. Replay Attacks

Mitigated by:

* Deduplication cache: (Source URI, Message ID) pairs.

* TTL: bounded hop count limits propagation.

* Timestamp option: implementations SHOULD include the Timestamp option and reject messages outside a deployment-specific freshness window.

9.6. Resource Exhaustion

Per-peer rate limiting (Section 7.5), maximum message size (65535 octets), and resolver cache capacity limits prevent resource exhaustion.

9.7. Semantic Resolution Gaming

When the SEM flag is used, the quality of results depends on the resolver’s scoring algorithm. Resolver implementations SHOULD incorporate anti-gaming measures such as trust signals and keyword-repetition dampening. These defenses are outside AIP’s scope and are defined by the resolver specification.

If the SEM flag is set but the SemQuery option is absent or malformed, implementations MUST treat the message as a protocol error and discard it. When the SIG flag is also set, the SemQuery option is covered by the signed message envelope, protecting provenance integrity.

9.8. Privacy

Agent:// URIs, resolver metadata, and message headers are visible to network participants. AIP does not provide payload confidentiality (AITP's responsibility) or metadata privacy. Applications requiring metadata privacy SHOULD use privacy-preserving lower-layer routing mechanisms.

10. IANA Considerations

This document requests the creation of four new IANA registries and the registration of a URI scheme.

10.1. URI Scheme Registration: agent

Per [RFC7595], the "agent" URI scheme is registered as follows:

Scheme name agent

Status Provisional

Applications/protocols that use this scheme Agent Internet Protocol (AIP), Agent Invocation Transport Protocol (AITP), Agent Name System (ANS).

Contact Jinke Song <ink@chatchat.space>

Change controller Jinke Song <ink@chatchat.space>

Reference [this document], Section 3

10.2. AIP Message Type Registry

Registry with 4-bit values (0-15). Registration policy: Specification Required. Initial values in Table 1.

10.3. AIP Protocol Number Registry

Registry with 8-bit values (0-255). Registration policy: Specification Required for 4-127, First Come First Served for 128-254. Value 255 is reserved for experimental use. Initial values in Table 2.

10.4. AIP Error Code Registry

Registry with 8-bit values (0-255). Value 0 reserved (MUST NOT use). Registration policy: Specification Required. Initial values in Table 5.

10.5. AIP Flag Bits Registry

Registry with 4 bit positions (0-3). Registration policy: Specification Required. Initial values in Table 3.

10.6. AIP Option Type Registry

Registry with 8-bit values (0-255). Registration policy: Specification Required for 0-127, First Come First Served for 128-254, value 255 reserved. Initial values in Table 4.

11. Implementation Status

Per [RFC7942], this section records known implementations.

11.1. ClawNet

Organization ChatChatTech

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

Description Go implementation of the full ANP suite. The AIP module (aip/ directory) implements the 16-octet fixed header, Protocol field demux, TLV options, local resolver interface, Ed25519 signature support, and deduplication cache. The reference implementation is being completed as part of the full ClawNet stack conforming to AIP, ANS, AITP, and ADP; protocol boundaries are exercised in integration tests across the stack.

Maturity Alpha

Coverage Implemented in current prototype:

* 16-octet header, Protocol demux, TLV option codec
* Local resolver interface, SEM flag delegation
* Deduplication cache, source URI validation, relay counting
* Message signature support (SIG flag codec)
* Max payload validation (65535)

Under integration; expected complete before final publication candidate:

* Ed25519 signature verification on receive

* Per-peer rate limiting

* Timestamp replay rejection

The items above affect robustness and enforcement behavior; they do not alter the wire format, message semantics, or protocol boundaries defined in this document.

Language Go

License Open source

Contact ink@chatchat.space

12. References

12.1. 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>.

[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>.

[RFC7595] Thaler, D., Ed., Hansen, T., and T. Hardie, "Guidelines and Registration Procedures for URI Schemes", BCP 35, RFC 7595, DOI 10.17487/RFC7595, June 2015, <https://www.rfc-editor.org/info/rfc7595>.

12.2. Informative References

[RFC0791] Postel, J., "Internet Protocol", STD 5, RFC 791, DOI 10.17487/RFC0791, September 1981, <https://www.rfc-editor.org/info/rfc791>.

[RFC1035] Mockapetris, P., "Domain Names - Implementation and Specification", STD 13, RFC 1035, DOI 10.17487/RFC1035, November 1987, <https://www.rfc-editor.org/info/rfc1035>.

[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>.

[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>.

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

[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. Example Resolver Scoring (Informative)

This appendix is entirely informative. It does not define any normative behavior and does not constrain AIP interoperability. Implementations are free to use any scoring strategy or none at all.

This appendix illustrates one possible scoring strategy that a semantic resolver could use. AIP does not mandate any particular scoring algorithm; this example is provided for implementers' reference.

The composite score for each candidate is computed as a weighted sum of five components:

```
score = w_text  * S_text      (BM25 text similarity, k1=1.2, b=0.75)
      + w_tag   * S_tag       (Jaccard tag overlap)
      + w_ns    * S_ns        (namespace match)
      + w_fresh * S_fresh     (1 / (1 + age_hours))
      + w_trust * S_trust     (trust score / max_trust)
```

Example weights: 0.4, 0.3, 0.05, 0.05, 0.2 (sum = 1.0).

Worked example with three resolver entries:

#	URI	Description	Skills	Age	Trust
1	agent://acme/fr-translator	French to English translation service	translation, french, english	2h	0.85
2	agent://babel/universal	Universal text translator, 50 languages	translation, multilingual	0.5h	0.92
3	agent://research/paper-search	Academic paper search and retrieval	research, search	1h	0.70

Table 6: Example Resolver Entries

Query: "translate French text", tags: ["translation", "french"], max_poi = 0.92.

Component scores:

	S_text	S_tag	S_ns	S_fresh	S_trust
Entry 1:	1.000	0.667	0	1/(1+2.0)=0.333	0.85/0.92=0.924
Entry 2:	0.664	0.333	0	1/(1+0.5)=0.667	0.92/0.92=1.000
Entry 3:	0.096	0.000	0	1/(1+1.0)=0.500	0.70/0.92=0.761

Composite (w_text=0.4, w_tag=0.3, w_ns=0.05, w_fresh=0.05, w_trust=0.2):

```
Entry 1: 0.4*1.000 + 0.3*0.667 + 0.05*0 + 0.05*0.333 + 0.2*0.924
       = 0.400 + 0.200 + 0.000 + 0.017 + 0.185 = 0.802

Entry 2: 0.4*0.664 + 0.3*0.333 + 0.05*0 + 0.05*0.667 + 0.2*1.000
       = 0.266 + 0.100 + 0.000 + 0.033 + 0.200 = 0.599

Entry 3: 0.4*0.096 + 0.3*0.000 + 0.05*0 + 0.05*0.500 + 0.2*0.761
       = 0.038 + 0.000 + 0.000 + 0.025 + 0.152 = 0.215
```

Result: Entry 1 (0.802) > Entry 2 (0.599) > Entry 3 (0.215). All above threshold (0.1).

Appendix B. Network Bootstrap (Informative)

This appendix is entirely informative. It describes a recommended bootstrap sequence for a new agent joining an ANP network for the first time. The steps below depend on companion protocols (AITP, ANS, ADP) and link-layer infrastructure; they do not alter AIP's wire format or processing rules.

B.1. Bootstrap Sequence

1. *Key Generation.*

   The agent generates an Ed25519 key pair. The public key derives the node's link-layer Peer ID. This step is entirely local and requires no network interaction.

2. *Seed Peer Discovery.*

   The agent obtains the Peer IDs and network addresses of one or more seed peers (also called bootstrap nodes). Seed information may come from a compiled-in bootstrap list, a DNS SRV record (see ANS [I-D.song-anp-ans], Appendix A), a well-known URI, an out-of-band configuration file, or manual input. At least one seed peer must be reachable for the remaining steps.

3. *Link-Layer Connection.*

   The agent's link-layer module (e.g., libp2p) establishes a transport connection to a seed peer using the seed's Peer ID and address. Upon connection, mutual peer authentication occurs at the link layer (e.g., Noise handshake in libp2p).

4. *DHT Join.*

   The agent joins the Kademlia DHT through the seed peer, populating its routing table. After this step, the agent can perform DHT get/put operations.

5. *Name Registration.*

   The agent constructs an ANS Name Record binding its chosen agent:// URI to its Peer ID, signs it, and submits it via ans.register (carried over AITP with Protocol = 1). The receiving peer validates the signature and stores the record. The agent also publishes the record to the DHT and announces it via GossipSub.

6. *Agent Card Publication.*

The agent constructs an ADP Agent Card describing its capabilities and publishes it via adp.advertise (carried over AITP, Protocol = 1). The Agent Card is also stored in the DHT under the /clawnet-profile/ namespace.

7. *Normal Operation.*

The agent's local resolver is now populated with at least one cached peer (the seed). AIP can send and receive datagrams. AITP associations can be established with any resolved peer. The agent is fully operational.

B.2. Minimal Bootstrap

Steps 5 and 6 are not required for baseline AIP operation. An agent that only needs to reach peers by known agent:// URIs can skip name registration and Agent Card publication and rely on DHT resolution of other agents' names.

Conversely, an agent that cannot reach any seed peer cannot join the network. AIP itself does not solve seed discovery; implementations MUST provide at least one bootstrap mechanism (compiled-in list, DNS, manual configuration) as part of their deployment profile.

Appendix C. Implementation Alignment Notes

This appendix documents the relationship between this specification and the reference implementation in the ClawNet aip/ directory.

Spec Concept	Go Symbol	Notes
Message Type	aip.MessageType	TypeData(0), TypeError(1), TypePing(2), TypePong(3).
Protocol Number	aip.Protocol	ProtoNone(0), ProtoAITP(1), ProtoANS(2), ProtoADP(3), ProtoExpt(255). (Go source still uses ProtoATP as a legacy alias.)
Flags	aip.Flags	FlagRly(0x1),

			FlagSem(0x2), FlagErr(0x4), FlagSig(0x8).
Error Code	aip.ErrorCode		Full 9-code set.
Fixed Header	fixedHeaderSize = 16		Byte layout per Figure 1.
Options	aip.Option, OptPad1..OptPriority		TLV encoding with 4-byte alignment.
Max payload	aip.MaxPayloadSize = 65535		Enforced in Marshal() and Unmarshal().
Local resolver	aip.LocalResolver (internal)		Implements the Resolver interface. BM25 scoring, LRU eviction. The Go type name is an implementation detail; AIP does not prescribe resolver internals.
SEM flag resolution	Module.Send()		Delegates to resolver's Discover() when FlagSem is set.
URI parsing	aip.ParseURI("agent://...")		Validates agent:// prefix, ident regex.

Table 7: Spec-Implementation Mapping

Appendix D. AIP in Isolation (Informative)

This appendix is entirely informative. It illustrates a minimal end-to-end message exchange using only AIP primitives, without expanding the internals of any companion protocol.

Scenario: Agent A wants to send an AITP payload to agent://translation/fr-ja. The local resolver already has a cached mapping for that URI.

1. Agent A calls SEND(source="agent://acme/requester", destination="agent://translation/fr-ja", protocol=1, payload=..., flags={SIG, RLY, ERR}).

2. AIP calls IS_LOCAL("agent://acme/requester") -> true. Source validation passes.

3. AIP calls RESOLVE("agent://translation/fr-ja") on the local resolver. The resolver returns (peer_id=0xABCD..., ok=true) from its cache. No distributed lookup is needed.

4. AIP constructs a DATA message: Version=1, Type=DATA, Protocol=AITP(1), TTL=8, Flags=0xD (SIG|ERR|RLY), Message ID=42, Payload Length=N. Source and destination URIs are wire-encoded (prefix stripped).

5. AIP computes the Ed25519 signature over the header, URIs, options, and payload; appends the 64-byte signature.

6. AIP calls TRANSMIT(peer_id=0xABCD..., serialized_bytes) on the link layer.

7. At the receiving node, the link layer calls DELIVER. AIP parses the header, verifies the signature against the public key bound to agent://acme/requester, checks deduplication, confirms IS_LOCAL("agent://translation/fr-ja") = true, and dispatches the payload to the Protocol=1 (AITP) handler.

At no point does AIP need to know how the resolver obtained its mapping, how AITP will process the payload, or what scoring strategy (if any) the resolver uses. AIP's responsibilities are: validate, resolve, construct, sign, transmit, and on the receive side: parse, verify, dedup, check TTL, deliver or relay.

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