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

J. Song
HKUST
M. Yuan
CUHK

25 March 2026

Agent Invocation Transport Protocol (AITP)
draft-song-anp-aitp-00

Abstract

The Agent Internet Protocol (AIP) provides best-effort, name-based datagram delivery between autonomous AI agents identified by agent:// URIs. The Agent Invocation Transport Protocol (AITP) is a message-framed invocation and streaming transport above AIP.

Unlike traditional host-to-host transports (TCP, QUIC) that operate on socket endpoints and byte streams, AITP is natively message-framed, method-aware, and association-aware: its header carries a method name as a first-class field, and its protocol data units are discrete requests, responses, stream chunks, and control segments exchanged between named agents. AITP is not a byte-stream transport; it is an agent-native invocation and streaming transport that carries method names, request/response correlation, flow control, and session semantics natively in its header — so that upper-layer agent protocols need not reinvent them. AITP intentionally includes a minimal generic invocation outcome space (four dispatch- level status codes); richer application semantics belong in the response body or upper-layer protocols.

Unlike transport bindings that adapt existing agent protocols to HTTP, gRPC, or MOQT, AITP defines a common invocation substrate directly above AIP, with association state, request-concurrency flow control, and method-aware framing as first-class transport concerns.

AITP is best understood as a common invocation substrate above AIP, not as a byte-stream transport analogue.

This document specifies the AITP segment format, segment types, status codes, flag bits, TLV options, association state machine, reliability engine, flow control, circuit-breaker mechanism, streaming, orderly close and abort, and interfaces to AIP below and application protocols above.

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.

Copyright Notice

Copyright (c) 2026 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (https://trustee.ietf.org/license-info) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document.

Table of Contents

1. Introduction

1.1. Problem Statement

AIP [I-D.song-anp-aip] provides a best-effort datagram service between agents identified by agent:// URIs: messages may be lost, duplicated, or reordered. Agent applications -- remote procedure calls, streaming inference, multi-turn conversations -- require reliable delivery with request-response correlation.

Existing host-to-host transports (TCP, QUIC) solve this problem for processes bound to IP addresses and ports. They provide reliable byte streams or datagrams between network endpoints, but they have no notion of agent:// addressing, association semantics between named agents, or message-level request/response framing. Using such transports directly forces every upper-layer agent protocol (task orchestration, knowledge replication, capability advertisement) to independently reinvent request IDs, timeouts, retransmission, flow control, and orderly close.

AITP bridges this gap. It is an agent-to-agent transport, not a host-to-host transport: its endpoints are agent:// URIs, its protocol data unit is a self-framed message (not a byte stream), and its association state tracks agent conversations rather than network connections. AITP accepts AIP datagrams carrying Protocol = 1 and provides upper-layer protocols with synchronous invocation, fire-and-forget messaging, and bidirectional streaming, all mediated through per-agent association state with request-concurrency flow control (counted in outstanding requests, not bytes) and circuit-breaker fault isolation.

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 (this document, 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 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.

AITP defines an experimental, lightweight invocation and streaming transport above AIP [I-D.song-anp-aip], intended for implementation and operational experimentation.

In one sentence: AIP delivers datagrams to agents; AITP provides a common invocation and streaming substrate on which those agents converse.

AITP is responsible for:

1. Association lifecycle -- establish, maintain, orderly close, and abort.

2. Request/response exchange with correlation by Request ID.

3. Fire-and-forget one-way messaging (NOACK flag).

4. Bidirectional streaming with FIN-based termination.

5. Status codes for transport-level outcomes and minimal handler-dispatch outcomes needed for generic interoperability.

6. Retransmission with exponential-backoff timeout and deduplication of retransmitted requests.

7. Receiver-advertised flow control via the Window field.

8. Per-association circuit breaker for fault isolation.

9. Extensible TLV options for timeout, sequencing, timestamps, signatures, and metadata.

AITP is NOT responsible for:

1. Naming, resolution, or agent discovery -- these belong to AIP and the Agent Name System (ANS).

2. Addressing and forwarding -- these belong to AIP.

3. End-to-end business semantics, task orchestration, or agent workflow -- these belong to upper-layer application protocols.

4. Economic settlement or credit systems.

5. Knowledge synchronisation or replication.

6. A general identity or authentication framework -- AITP provides optional segment-level signatures (SIGNED flag) and defers to AIP and deployment-context mechanisms for identity binding.

7. Payload confidentiality or end-to-end encryption -- these are application-layer responsibilities.

8. Message fragmentation -- AITP relies on AIP's maximum message size (65535 octets).

Accordingly, this document does not specify a complete agent runtime, workflow engine, business protocol family, or deployment profile; it defines only the common invocation and streaming substrate shared by such systems.

Because AITP carries a Method name as a first-class header field and defines minimal handler-dispatch status codes, it is not a pure transport in the TCP/QUIC sense. It is an agent-native invocation transport that unifies method dispatch, request/response correlation, retry, flow control, and session management into a shared substrate so that multiple upper-layer agent protocols (ADP, task orchestration, capability advertisement, knowledge sync, and future agent message families) need not independently reinvent these mechanics. AITP is best understood as a common invocation substrate above AIP, rather than as a byte-stream transport analogue.

1.3. Relationship to Existing Transport Protocols

A natural question is: "Why not use QUIC streams with gRPC, or libp2p streams with a custom RPC layer?" This section explains why AITP occupies a distinct position in the protocol stack.

1.3.1. Different Abstraction Objects

TCP and QUIC are host-to-host (or process-to-process) transports. Their endpoints are IP addresses and ports; their primary abstraction is the connection or stream between network endpoints. They do not understand agent:// URIs, resolver-assisted delivery, or agent association semantics.

AITP is an agent-to-agent transport. Its endpoints are agent:// URIs; its primary abstraction is the association between named agents. An association carries not just connectivity state, but also request correlation, flow control windows counted in outstanding requests (not bytes), and a circuit breaker reflecting the remote agent's operational health -- concepts that have no direct counterpart in traditional transport protocols.

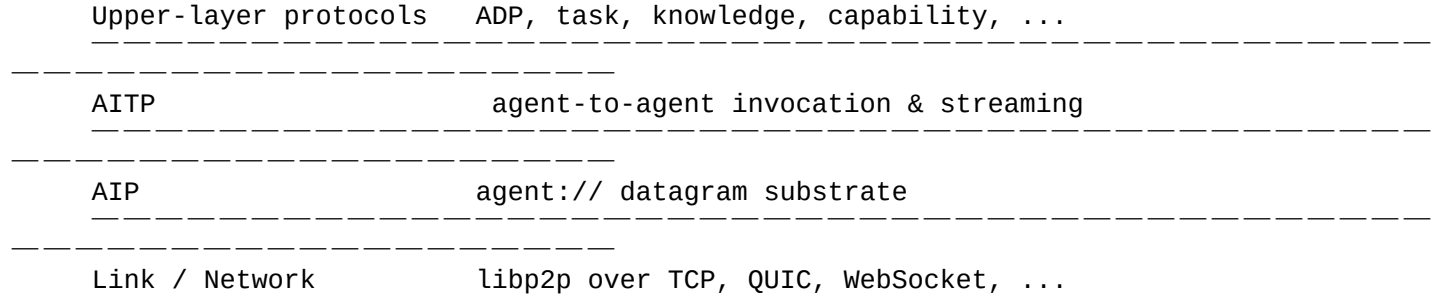
1.3.2. Message-Framed, Not Byte-Stream

TCP provides an unstructured byte stream; applications must impose their own framing. QUIC improves on this with multiple streams but remains byte-oriented within each stream.

AITP is natively message-framed. Every segment is a self-contained protocol data unit with a typed header, request ID, method name, status code, and body. The transport itself understands "this is a request", "this is a response", "this is a stream chunk", and "this is a control message" -- eliminating the need for upper-layer protocols to define their own framing and multiplexing.

1.3.3. Complementary Layering

AITP does not compete with TCP or QUIC for the same layer. The correct relationship is:



Traditional transports provide network reachability and basic reliable delivery at the link/network layer. AIP provides agent-oriented, name-based datagram delivery above that. AITP provides reliable invocation and streaming semantics above AIP. Each layer has a distinct job; removing any one forces its responsibilities onto an adjacent layer.

AITP does not replace lower-layer transports such as TCP or QUIC; it defines a reusable invocation substrate for agent://-addressed communication above AIP.

1.4. Positioning Among Agent Communication Approaches

The preceding section distinguishes AITP from host-to-host transports. This section clarifies its relationship to agent-oriented communication approaches that operate at different layers of the stack.

Several agent protocol families (e.g., A2A) define application-level messaging semantics — task lifecycle, capability exchange, content negotiation — and then bind those semantics onto existing transports such as HTTP or gRPC. AITP does not define application semantics. It occupies a lower position: a reusable invocation and streaming substrate above AIP, designed to be shared by multiple upper-layer agent protocols rather than serving one particular protocol family.

Client-server agent transports such as MCP's stdio and Streamable HTTP provide carriage for JSON-RPC exchanges between a client and a server, inheriting session and flow-control semantics from the underlying transport or runtime environment. AITP targets symmetric agent-to-agent exchanges between agent://-identified endpoints; association state and request-concurrency control are defined within the protocol itself rather than inherited from an underlying HTTP or process-level session.

Deployment platforms and runtime messaging fabrics may provide routing, streaming, or orchestration facilities for agents. AITP does not attempt to standardize such systems. It specifies a protocol-layer invocation substrate that can be implemented independently of any particular runtime while remaining reusable across agent-oriented application protocols built above AIP.

1.5. Interoperability Model

This section summarizes the minimum requirements for implementing an AITP endpoint, achieving two-party interoperability, and distinguishing core protocol behaviour from optimization profiles.

1.5.1. Minimum Endpoint Requirements

A conforming AITP endpoint requires:

1. An AIP module providing SEND and RECEIVE primitives with Protocol = 1.

2. An association table keyed by (localURI, remoteURI) that tracks lifecycle state, flow control window, and circuit-breaker state.

3. Request ID tracking for correlation and deduplication.

4. A method-dispatch surface capable of mapping Method values to local handlers.

1.5.2. Two-Party Interoperability Conditions

Two implementations interoperate when they share:

1. The same AITP version (currently 1).

2. Agreed semantics for the flags, status codes, and option types they exchange.

3. Compatible compression and signature expectations, if the COMPR or SIGNED flags are used.

1.5.3. Core versus Profile

The following are optimization profiles or implementation choices, not core protocol requirements:

* Lazy association establishment (core baseline is the explicit INIT handshake).
* Compression algorithm selection (COMPR flag).
* Operational telemetry and monitoring counters.
* Deployment-specific defaults for window size, retransmission parameters, and circuit-breaker thresholds.

1.6. Design Rationale

AITP is justified when the following conditions hold:

1. *Multiple upper-layer protocols share transport needs.*

   If ADP, task orchestration, knowledge sync, and capability advertisement each need method dispatch, request/response correlation, retransmission, timeout, request-concurrency flow control, and orderly close, factoring these into a shared invocation transport eliminates redundant specification and implementation.

2. *AIP remains a narrow waist.*

   If reliable delivery, association state, and request/response semantics were pushed down into AIP, AIP would lose its simplicity as a best-effort datagram layer. AITP exists precisely so that AIP can stay thin.

3. *Agent-native addressing and conversation semantics are valued as an independent standard layer.*

   If a deployment is willing to fully delegate transport to an existing RPC framework (e.g., gRPC over QUIC), AITP is not required. AITP is valuable when the community seeks a self-contained, agent-native invocation transport specification that does not depend on any specific lower-layer RPC stack.

A practical test: if removing AITP would cause every upper-layer agent protocol to independently redefine method dispatch, request IDs, retry policies, timeout handling, flow control, and session close/abort, then AITP earns its place in the architecture.

Method is a first-class header field. Upper-layer agent protocols rely on it for dispatch; AITP provides the invocation framing so that they do not each define their own. This is what makes AITP an invocation substrate rather than a generic message bus, and is the primary reason its flow control counts outstanding requests rather than bytes.

Because AITP is invocation-oriented rather than byte-stream-oriented, its flow control regulates concurrent outstanding requests admitted by the remote agent, not raw octet volume on the wire.

1.7. Assumptions

AITP assumes:

1. An AIP module capable of sending and receiving datagrams with Protocol = 1. AITP does not require any specific AIP implementation beyond the SEND/RECEIVE primitives defined in [I-D.song-anp-aip].

2. Agent:// URIs are stable identifiers for the duration of an association. If an agent migrates, existing associations MAY be invalidated.

3. AITP MAY be used by companion upper-layer protocols such as capability advertisement (ADP), task orchestration, or knowledge synchronisation, but does not depend on any specific upper-layer protocol for baseline operation.

4. The receiving side exposes a method-dispatch surface capable of mapping Method values to local handlers. AITP does not prescribe how such handlers are represented internally.

1.8. 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

Segment The protocol data unit of AITP. A self-contained unit carrying a fixed header, method name, TLV options, and body. Segments are carried as AIP datagram payloads with Protocol = 1.

Association A stateful transport-layer relationship between two

agents identified by their agent:// URIs. An association tracks lifecycle state, flow control windows, and circuit-breaker state.

Request ID A 32-bit sender-assigned identifier used to correlate requests with responses and to detect duplicates. Request IDs MUST be unique among outstanding requests within a single association. A stream is a long-lived request context identified by its opening Request ID.

Method A UTF-8 string (up to 255 octets) identifying the operation to be invoked on the remote agent. Method is a first-class header field; upper-layer protocols rely on it for dispatch without defining their own invocation framing.

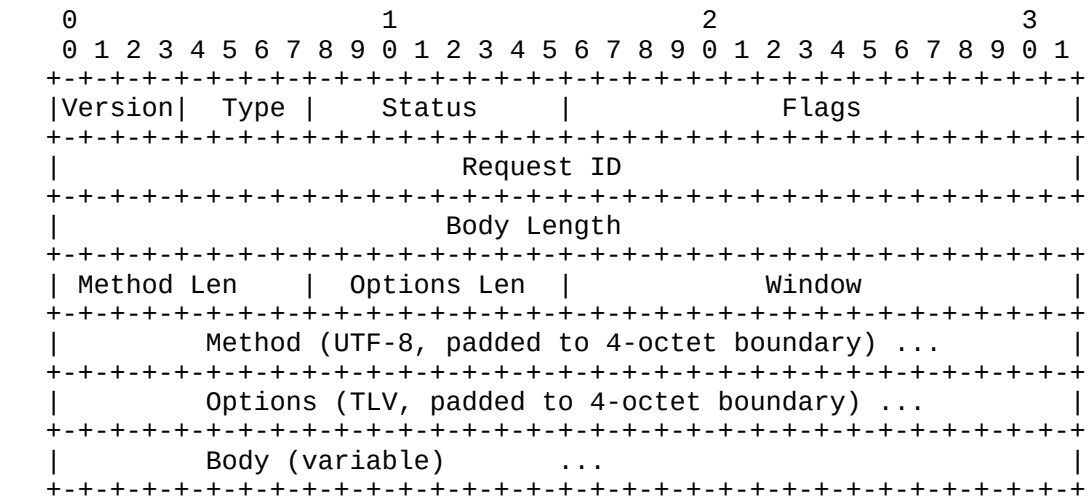
Flow Control Window A 16-bit value advertised by the receiver indicating the maximum number of concurrent in-flight requests it is willing to accept from the sender.

Circuit Breaker A per-association fault-isolation mechanism that prevents sending requests to a peer that is experiencing persistent failures.

3. AITP Segment Format

AITP segments are carried as AIP datagram payloads with Protocol = 1. A segment consists of a 16-octet fixed header, a variable-length method name (padded to a 4-octet boundary), a variable-length TLV options region (padded to a 4-octet boundary), and a variable-length body.

3.1. Header Structure



Note: one tick mark represents one bit position.

Figure 1: AITP Segment Header (16 octets fixed)

3.2. Field Definitions

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

Type (4 bits) The segment type:

Value	Name	Description
0	REQUEST	Invocation request. Carries a method name and request body.
1	RESPONSE	Invocation response. Carries a status code and response body.
2	STREAM	Streaming data chunk. Part of a bidirectional stream identified by Request ID.
3	CONTROL	Association lifecycle control. Exactly one of INIT, FIN, or RST MUST be set; invalid combinations MUST be rejected.
4-15		Reserved for future use.

Table 1: AITP Segment Types

Implementations MUST silently discard segments with unrecognized Type values.

Status (8 bits) Result status code. Primarily meaningful in RESPONSE segments; in REQUEST and STREAM segments this field SHOULD be set to 0 (OK).

AITP status codes span two tiers. Transport-processing codes (0-1, 3-4, 6, 9) reflect outcomes that any message-framed transport must report. Handler-dispatch codes (2, 5, 7-8) report minimal method-invocation outcomes needed for generic interoperability; they allow a sender to distinguish "method not found" from "transport error" without requiring application-layer parsing. Dispatch-level status codes are intentionally minimal and exist only to support generic upper-layer interoperability; richer application semantics belong in the response body or upper-layer protocols, not in the Status field.

Value	Name	Tier	Description
0	OK	Transport	Success.
1	ERROR	Transport	Generic error.
2	NOT_FOUND	Dispatch	Method not

			registered on the remote agent.
3	TIMEOUT	Transport	Request timed out (generated locally by the reliability engine).
4	BUSY	Transport	Remote agent is overloaded; retry later.
5	UNAUTHORIZED	Dispatch	Caller lacks permission for the requested method.
6	INVALID_REQUEST	Transport	Malformed request (bad parameters, missing method, etc.).
7	INTERNAL_ERROR	Dispatch	Unspecified internal processing error on the remote agent.
8	NOT_IMPLEMENTED	Dispatch	Method recognized but not yet implemented.
9	SERVICE_SHUTDOWN	Transport	Remote agent is shutting down gracefully.
10-255			Available for future assignment (IANA registry).

Table 2: AITP Status Codes

In summary, the four dispatch codes (NOT_FOUND, UNAUTHORIZED, INTERNAL_ERROR, NOT_IMPLEMENTED) form AITP's minimal generic invocation outcome space. They exist so that any upper-layer protocol can interpret a basic method-dispatch result without parsing the response body. These four outcomes are sufficient to distinguish transport success from basic invocation failure classes without committing AITP to any application-specific error

taxonomy; they support interoperable invocation behavior across heterogeneous upper-layer protocols without encoding business-level semantics. Richer business semantics — task state, payment results, capability negotiation outcomes — MUST be carried in the response body or defined by upper-layer protocols, not encoded as AITP status codes.

Flags (16 bits) A bitmask of control flags:

Bit	Value	Name	Description
0	0x0001	ACK	Acknowledgment. Set in RESPONSE segments and CONTROL acknowledgments.
1	0x0002	FIN	End of stream or association close. Signals graceful termination.
2	0x0004	INIT	Association initialization request.
3	0x0008	RST	Reset. Immediately aborts the association.
4	0x0010	SEQ	Sequence number present in options (for ordered delivery within a stream).
5	0x0020	NOACK	No acknowledgment requested. Used for fire-and-forget semantics.
6	0x0040	COMPR	Body is compressed. Compression algorithm is deployment-specific. Peers MUST either agree on the compression format out of band or reject compressed segments they cannot decode.
7	0x0080	SIGNED	Segment carries a cryptographic signature in options.
8-13			Reserved for future use.
14	0x4000	CBOPEN	Circuit breaker: half-open probe request.
15	0x8000	CBTRIP	Circuit breaker: tripped notification to peer.

Table 3: AITP Flag Bits

Request ID (32 bits) A sender-assigned identifier for request-

response correlation and deduplication. Each new request MUST use a unique Request ID among outstanding requests within the same association. RESPONSE segments MUST echo the Request ID of the corresponding REQUEST.

A stream-opening STREAM segment consumes a Request ID for the lifetime of the stream. The ID MUST NOT be reused for another request or stream within the same association until the stream has terminated (FIN exchanged or RST received).

Body Length (32 bits) The length in octets of the body: the request body in REQUEST segments, or the response body in RESPONSE segments. A Body Length of 0 is valid (e.g., for CONTROL segments or parameterless requests).

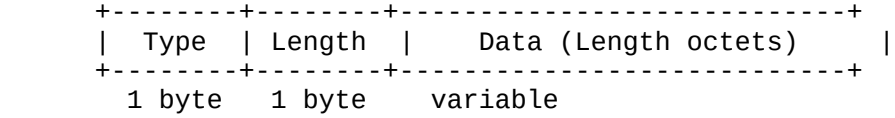
Method Len (8 bits) The length in octets of the UTF-8 method name. Maximum 255. A value of 0 is permitted for CONTROL segments that carry no method. The method field is padded to a 4-octet boundary with zero-valued octets.

Options Len (8 bits) The total length in octets of the TLV options region, including any padding. Maximum 255. The options region MUST be padded to a 4-octet boundary.

Window (16 bits) The sender’s advertised receive window: the number of concurrent in-flight requests the sender is willing to accept. The receiver uses this value for flow control (see Section 6). The initial value is deployment-specific; implementations MAY advertise any value from 1 to 65535.

3.3. TLV Options

The options region carries zero or more Type-Length-Value (TLV) encoded options, following the same encoding as AIP options:



Type	Name	Length	Description
1	Timeout	4	Request timeout in milliseconds (uint32, big-endian). Included in REQUEST segments to advise the receiver of the caller's deadline.
2	SeqNum	4	Sequence number (uint32, big-endian). Used with the SEQ flag for ordered delivery within a stream.
3	AckNum	4	Acknowledgment number (uint32, big-endian). Acknowledges receipt of a specific sequence number.
4	Timestamp	8	UTC Unix timestamp in microseconds (uint64, big-endian). Used for latency measurement.
5	Signature	variable	Cryptographic signature over the segment. Used with the SIGNED flag.
6	Metadata	variable	Key-value metadata (encoding is application-defined).
7-127			Available for future assignment (IANA).
128-254			Available for experimental / private use.
255			Reserved.

Table 4: Standard AITP Options

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

4. Association State Machine

An association is a stateful transport-layer relationship between two agents, identified by the (localURI, remoteURI) pair. Each association tracks lifecycle state, flow control parameters, and circuit-breaker state.

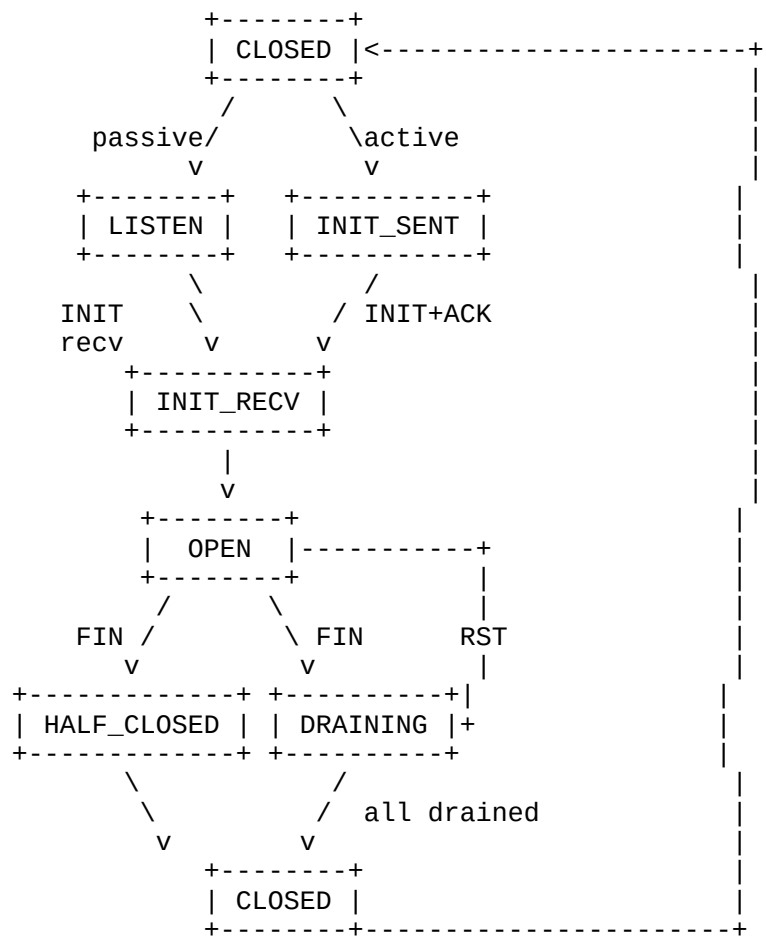
4.1. Association States

Value	Name	Description
0	CLOSED	No association exists. Initial and terminal state.
1	LISTEN	Waiting for an incoming INIT from a remote peer.
2	INIT_SENT	INIT sent; awaiting INIT+ACK from the remote peer.
3	INIT_RECV	INIT received; INIT+ACK sent; awaiting first data exchange.
4	OPEN	Association fully established. Data exchange is permitted.
5	HALF_CLOSED	Local side has sent or received FIN; no new requests accepted, but in-flight responses may still arrive.
6	DRAINING	Both sides have exchanged FIN; draining in-flight segments before transitioning to CLOSED.

Table 5: AITP Association States

4.2. State Transitions

The following transitions are valid. Any other transition MUST be rejected as a protocol error.



Valid transitions from each state:

CLOSED LISTEN (passive open) or INIT_SENT (active open).

LISTEN INIT_RECV (INIT received) or CLOSED (abort).

INIT_SENT OPEN (INIT+ACK received) or CLOSED (timeout/abort).

INIT_RECV OPEN (handshake complete) or CLOSED (abort).

OPEN HALF_CLOSED (FIN sent/received), DRAINING, or CLOSED (RST).

HALF_CLOSED DRAINING or CLOSED.

DRAINING CLOSED (all segments drained).

4.3. Lazy Association Establishment

The explicit INIT/INIT+ACK handshake defined in Section 4.2 is the fully specified, interoperable baseline. Implementations that require mutual confirmation before data exchange MUST use this handshake.

LISTEN exists to define the passive side of the explicit interoperable baseline; implementations that prefer lazy outbound establishment still need to accept incoming explicit INIT exchanges.

As an optimization, implementations MAY use lazy association establishment: when a module needs to send a request to a peer with no existing association, it creates an association and transitions directly through INIT_SENT to OPEN without an explicit on-wire handshake. This is an optimization profile for deployments whose lower layer (e.g., libp2p with peer authentication) already guarantees peer identity, making the INIT round-trip redundant.

Implementations MUST document whether they use the explicit handshake or lazy establishment as their default mode, and MUST accept incoming INIT segments regardless of which mode they prefer for outgoing associations. Interoperability claims for AITP MUST be evaluated against the explicit- handshake baseline.

5. Reliability Engine

AITP provides reliability through ACK-based tracking, retransmission with exponential backoff, and deduplication. The reliability engine operates on a per-request basis: each outgoing REQUEST that does not carry the NOACK flag is tracked until a matching RESPONSE arrives or the request times out.

5.1. ACK Tracking

When a REQUEST segment is sent (without the NOACK flag), the sender registers the Request ID in a pending-request table. When a RESPONSE segment arrives with a matching Request ID, the pending entry is removed and the response is delivered to the caller.

If the NOACK flag is set, the sender does not track the request and does not expect a response. The receiver SHOULD NOT send a response for NOACK requests, but MAY still invoke the method handler for side effects.

5.2. Retransmission

If no RESPONSE arrives within the retransmission timeout, the sender retransmits the REQUEST segment. The retransmission timeout uses exponential backoff:

```
timeout(n) = InitialTimeout * BackoffFactor^n
```

where n = retransmission attempt number (0-based)

After MaxRetries retransmissions without a response, the reliability engine generates a local TIMEOUT response (Status = TIMEOUT) and delivers it to the caller. Deployment-specific defaults are expected for InitialTimeout, MaxRetries, and BackoffFactor; implementations SHOULD document their default values.

5.3. Deduplication

Retransmitted REQUEST segments carry the same Request ID as the original. The receiver MUST maintain a deduplication cache keyed by Request ID. If an incoming REQUEST matches a recently-seen Request ID, the receiver MUST silently discard it (the response to the original request will serve as the acknowledgment).

Implementations MUST bound the deduplication cache size and entry lifetime to prevent unbounded resource consumption; deployment-specific defaults are expected.

6. Flow Control

AITP flow control is request-concurrency based, not byte-volume based. This is a key departure from traditional transports: the Window field counts outstanding REQUEST operations, not octets. The peer MUST track the remote window and MUST NOT have more in-flight requests than the advertised window allows.

Before sending a REQUEST, the sender checks that its in-flight count is below the peer's advertised window and that the circuit breaker allows sending (see Section 7). If either check fails, the send MUST be rejected with an appropriate error to the upper layer.

When a RESPONSE arrives, the sender decrements its in-flight count and updates the peer's window from the RESPONSE's Window field. This allows the receiver to dynamically adjust its capacity.

The initial window size is deployment-specific. Implementations MAY advertise any value from 1 to 65535; a value of 16 is a reasonable starting point for general-purpose agents.

The Window field governs concurrent in-flight REQUEST operations only. STREAM chunks are subject to stream- specific bounded buffering and back-pressure (see Section 8.2), not request-window accounting. An active stream consumes one Request ID slot for its lifetime but does not consume additional window capacity for subsequent STREAM chunks.

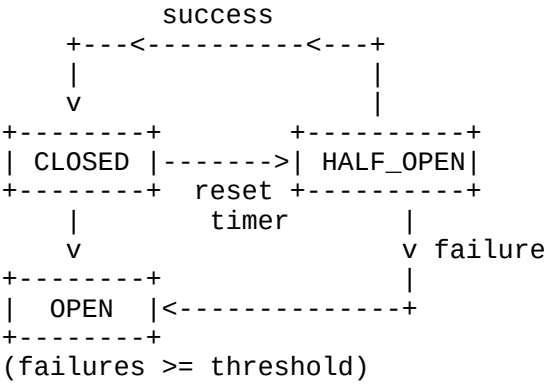
7. Circuit Breaker

Each association maintains a per-peer circuit breaker to prevent cascading failures when a remote agent is unavailable or consistently failing. The circuit breaker has three states:

Value	Name	Description
0	CLOSED	Normal operation. Requests pass through.
1	OPEN	Tripped. All requests are rejected immediately without being sent.
2	HALF_OPEN	Probing. One request is allowed through to test recovery.

Table 6: Circuit Breaker States

7.1. Circuit Breaker Transitions



CLOSED → OPEN: When the consecutive failure count reaches the failure threshold, the circuit breaker transitions to OPEN. Implementations SHOULD use deployment-specific defaults for the threshold.

OPEN → HALF_OPEN: After a reset timeout elapses since the last failure, the circuit breaker transitions to HALF_OPEN, allowing one probe request.

HALF_OPEN → CLOSED: If the probe request succeeds, the circuit breaker resets to CLOSED and the failure count is cleared.

HALF_OPEN → OPEN: If the probe request fails, the circuit breaker returns to OPEN.

7.2. Circuit Breaker Flag Bits

The CBOPEN flag (bit 14) MAY be set on a probe request to inform the remote peer that this is a circuit-breaker probe. The CBTRIP flag (bit 15) MAY be set by a peer to proactively notify the sender that it should trip its circuit breaker. These flags are advisory; implementations are not required to honor them.

8. Streaming

AITP supports bidirectional streaming for long-running exchanges such as inference generation, file transfer, or multi-turn conversations. A stream is identified by its Request ID and uses STREAM segments (Type = 2).

8.1. Stream Lifecycle

1. *Initiation.*

   The initiator sends the first STREAM segment with a fresh Request ID and a method name. If the receiver has a registered stream handler for that method, a new stream is created. If not, the receiver returns a NOT_FOUND RESPONSE.

2. *Data exchange.*

   Both sides send STREAM segments with the same Request ID. Each segment carries a body chunk. The SEQ flag and SeqNum option MAY be used for ordered delivery within a stream.

3. *Termination.*

   Either side sends a STREAM segment with the FIN flag to signal end of stream. Upon receiving FIN, the receiving side closes its chunk channel. Both sides SHOULD release stream resources after FIN.

8.2. Back-Pressure

Stream receivers maintain a bounded buffer for incoming chunks. If the buffer is full, incoming STREAM segments MAY be dropped (back-pressure). The sender SHOULD use the association's flow control window and application-level acknowledgments to regulate sending rate.

8.3. Large Payload Guidance

AIP limits each datagram payload to 65535 octets. Upper-layer protocols whose response objects may exceed this limit SHOULD use one of the following strategies:

1. *Streaming.*

   Use an AITP STREAM exchange to deliver the response incrementally. This is appropriate for variable-size payloads such as large Agent Cards, discovery result sets, or inference outputs. Each STREAM chunk carries a portion of the response; the receiver reassembles them in SeqNum order (when the SEQ flag is used) or in arrival order.

2. *Application-level pagination.*

   Define "offset" and "limit" parameters in the REQUEST body so that the caller can page through a large result set with multiple REQUEST/RESPONSE round trips. This is appropriate for enumeration operations (e.g., ans.lookup, adp.discover) where the total result count may be large but each page fits in a single datagram.

Upper-layer methods that are expected to return large payloads SHOULD document which strategy they use and whether the method's tool descriptor sets streaming = true.

9. Segment Processing

This section defines AITP's complete processing model. The send and receive paths described below depend only on AIP's SEND/RECEIVE primitives and the local association/reliability state. No knowledge of upper-layer application protocols is required to implement these paths.

9.1. Sending a Request

When the upper layer invokes SEND-REQUEST or SEND-ONEWAY:

1. *Ensure association.*

   Look up or create an association for the destination agent:// URI. If lazy establishment is used, transition directly to OPEN. If explicit handshake is required, send CONTROL+INIT and wait for CONTROL+INIT+ACK.

2. *Check flow control.*

   Verify that the in-flight count is below the peer's advertised window and the circuit breaker allows sending. If not, return an error to the caller.

3. *Construct the segment.*

   Build the 16-octet fixed header with Version = 1, Type = REQUEST (or STREAM), Status = OK, appropriate Flags, a fresh Request ID, Body Length, Method Len, Options Len, and the local receive Window. Serialize method, options, and body.

4. *Register for tracking (if not NOACK).*

   Add the Request ID to the reliability engine's pending table and start the retransmission timer.

5. *Transmit via AIP.*

   Call AIP's SEND with source = local agent:// URI, destination = remote agent:// URI, protocol = 1 (AITP), and the serialized segment as payload.

9.2. Receiving a Segment

When AIP delivers a datagram with Protocol = 1 to the AITP module:

1. *Deserialize.*

   Decode the fixed header. If the Version is unrecognized or the data is truncated, discard.

2. *Update association.*

   Look up or create the association for the source URI. Update the last-seen timestamp. If the segment carries a non-zero Window, update the peer's advertised window.

3. *Dispatch by Type.*

REQUEST: Check deduplication. If duplicate, discard. Look up the method handler. If not found, send a RESPONSE with Status = NOT_FOUND. Otherwise, invoke the handler asynchronously. When the handler completes, send a RESPONSE with the handler's status and response body (unless NOACK).

RESPONSE: Deliver to the reliability engine, which matches by Request ID and wakes the waiting caller. Update flow control: decrement in-flight count and record circuit-breaker success/ failure based on the status code.

STREAM: If the Request ID matches an active stream, deliver the chunk to the stream's buffer. If FIN is set, close the stream's input channel. If no stream exists, check for a stream handler and create a new stream. If no handler, send RESPONSE with NOT_FOUND.

CONTROL: Verify that exactly one of INIT, FIN, or RST is set. Invalid CONTROL flag combinations MUST NOT create, mutate, or tear down association state; discard the segment. INIT → transition association to INIT_RECV then OPEN, reply with CONTROL+INIT+ACK; FIN → transition to HALF_CLOSED then DRAINING, reply with CONTROL+FIN+ACK, then close; RST → transition immediately to CLOSED.

10. Interfaces

10.1. Upper Layer Interface (Application Protocols)

AITP provides the following abstract service primitives to upper-layer protocols. These are protocol-level abstractions; actual API bindings are implementation- defined.

SEND-REQUEST(destination, method, request-body, timeout) -> (status, response-body) Send a request and wait for a response.

- *destination* -- agent:// URI of the remote agent.
- *method* -- UTF-8 method name (max 255 octets).
- *request-body* -- opaque payload bytes.
- *timeout* -- optional request deadline.

Returns: status code and response body, or a transport error.

SEND-ONEWAY(destination, method, request-body) -> error Send a one-

way message with no response expected. Sets the NOACK flag. Returns only a send error, not a remote result.

OPEN-STREAM(destination, method) -> stream Initiate a bidirectional stream. Returns a stream handle through which the caller sends and receives STREAM chunks.

REGISTER-HANDLER(method, handler) Register a handler that will be invoked when an incoming REQUEST arrives for the given method.

REGISTER-STREAM-HANDLER(method, handler) Register a handler that will be invoked when an incoming STREAM segment arrives for the given method with no active stream.

Monitoring and statistics primitives (e.g., counters for requests sent, streams active) are implementation concerns and are not specified by this document. Deployment-specific counters, tracing hooks, and operational telemetry are expected to be defined by implementation profiles rather than by this protocol specification.

10.2. Lower Layer Interface (AIP)

AITP depends on AIP for datagram delivery. The interface consists of:

AIP.SEND(source, destination, protocol=1, payload) Sends an AITP segment as an AIP datagram. AITP always uses Protocol = 1.

AIP.RECEIVE callback (Protocol = 1) AIP dispatches incoming datagrams with Protocol = 1 to AITP’s HandleSegment entry point.

Note: AIP assigns Protocol values 2 (ANS) and 3 (ADP) for potential future direct-datagram exchanges that bypass AITP. In normal operation, ANS and ADP methods are carried as AITP segments with Protocol = 1. See AIP [I-D.song-anp-aip], Section 4.

11. Security Considerations

11.1. Segment Authentication

AITP defines the SIGNED flag and Signature option for segment- level authentication. When the SIGNED flag is set, implementations SHOULD verify the signature before processing the segment. The signature algorithm and key binding are defined by the deployment context; AITP does not mandate a specific algorithm.

AITP segment signatures protect invocation-layer provenance and segment integrity, while AIP signatures protect the enclosing datagram. Deployments MAY use either layer or both, depending on their threat model, and SHOULD use at least one.

11.2. Replay Attacks

AITP's deduplication cache (Section 5.3) provides protection against replayed REQUEST segments. The Timestamp option provides an additional freshness signal. Implementations SHOULD reject segments with timestamps outside a deployment-specific freshness window.

11.3. Resource Exhaustion

Flow control (Section 6) limits the number of concurrent in-flight requests per association. The circuit breaker (Section 7) prevents runaway retries to failing peers. Implementations MUST bound the deduplication cache, pending-request table, and stream buffer sizes to prevent unbounded memory growth.

11.4. Method Authorization

AITP provides the UNAUTHORIZED status code (5) for method- level access control. The authorization mechanism is application-defined; AITP does not prescribe any particular access control model. Implementations SHOULD document their authorization strategy.

11.5. Privacy

Method names and request/response bodies are visible to network participants that can observe AIP datagrams. AITP does not provide payload confidentiality. Applications requiring confidentiality SHOULD encrypt payloads at the application layer or use privacy-preserving lower-layer routing mechanisms.

11.6. Association State Attacks

Because CONTROL segments (INIT, FIN, RST) can create, close, or abort associations, unauthenticated control segments can induce association churn or exhaust the association state table.

Deployments SHOULD authenticate peers (via AIP SIG, AITP SIGNED, or lower-layer mechanisms) before creating persistent association state. Implementations SHOULD rate-limit new association creation per source URI and MUST bound the total number of concurrent associations to prevent unbounded state growth.

11.7. Dispatch-Level Information Leakage

The dispatch-level status codes NOT_FOUND (2), UNAUTHORIZED (5), and NOT_IMPLEMENTED (8) reveal information about the methods registered on the remote agent. An adversary can enumerate supported methods by sending probe requests and observing which status code is returned.

Deployments concerned with method enumeration MAY coarsen dispatch-level status codes into a generic ERROR or UNAUTHORIZED response for unrecognized or unauthorized callers, at the cost of reduced diagnostic fidelity for legitimate clients.

12. IANA Considerations

This document requests the creation of four new IANA registries.

12.1. AITP Segment Type Registry

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

12.2. AITP Status Code Registry

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

12.3. AITP Flag Bits Registry

Registry with 16 bit positions (0-15). Registration policy: Specification Required. Initial values in Table 3.

12.4. AITP 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.

13. Implementation Status

Per [RFC7942], this section records known implementations.

13.1. ClawNet

Organization ChatChatTech

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

Description Go implementation of the full ANP suite. The AITP module (atp/ directory) implements the 16-octet fixed header, segment type demux, TLV options, association state machine, reliability engine with exponential-backoff retransmission, receiver-advertised flow control, per-association circuit breaker, bidirectional streaming, and the full 10-code status code set. 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:

* Segment codec (16-octet header, type demux, TLV options)
* Association state machine (7 states, validated transitions)
* Reliability engine (ACK tracking, exponential-backoff retransmission, dedup)
* Receiver-advertised flow control (Window)
* Per-association circuit breaker (CLOSED/OPEN/HALF_OPEN)
* Synchronous invocation and fire-and-forget
* Bidirectional streaming with FIN termination
* Lazy association establishment

Under integration:

* Segment-level signature verification (SIGNED flag)
* Body compression (COMPR flag)
* Explicit INIT/FIN handshake integration tests

The items under integration affect robustness and efficiency only; they do not alter the wire format, segment semantics, or protocol boundaries defined in this document.

Language Go

License Open source

Contact ink@chatchat.space

14. References

14.1. 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>.

[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>.

14.2. 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>.

[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. Implementation Alignment Notes

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

Spec Concept	Go Symbol	Notes
Segment Type	atp.SegmentType	TypeRequest(0), TypeResponse(1), TypeStream(2), TypeControl(3).
Status Code	atp.StatusCode	Full 10-code set: StatusOK(0) through StatusServiceShutdown(9).
Flags	atp.Flags	FlagACK(0x1), FlagFIN(0x2), FlagINIT(0x4), FlagRST(0x8), FlagSEQ(0x10), FlagNOACK(0x20), FlagCOMPR(0x40), FlagSIGNED(0x80), FlagCBOPEN(0x4000), FlagCBTRIP(0x8000).
Fixed Header	fixedHeaderSize = 16	Byte layout per Figure 1.
Options	atp.Option, OptTimeout..OptMetadata	TLV encoding with 4-byte alignment.
Association	atp.Association	7-state FSM per Table 5.
Circuit Breaker	atp.CBState	CBClosed(0), CBOpen(1), CBHalfOpen(2).
Reliability	atp.Reliability	Track(), Resolve(), IsDuplicate(), retransmitLoop().
Flow Control	Association.AcquireSend/ ReleaseSend	Window-based concurrency limiting.
Module	atp.Module	SEND-REQUEST → Call(), SEND-ONEWAY → FireAndForget(), HandleSegment(), REGISTER-HANDLER → Handle(), REGISTER-

		STREAM-HANDLER → HandleStream().	
Stream	atp.Stream	Recv(), Send(), Close() with FIN termination.	
AIP Interface	atp.AIPSender	SendRaw(ctx, src, dst, proto, payload).	

Table 7: 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