RFC 0.1: Semantic Intent Negotiation Protocol (SINP)

Specification Document
Version: 0.1 (Verified Draft)

December 31, 2025

Abstract

This document specifies the Semantic Intent Negotiation Protocol (SINP), an application-layer protocol designed to replace deterministic address-routing with semantic negotiation. It defines a rigorous state machine where endpoints exchange *Intent Descriptions* and resolve execution through mutual *Confidence* scores. This version (0.1) includes explicit wire formats, cryptographic canonicalization rules, decision-theoretic thresholds, and failure modes for probabilistic interpreters (LLMs).

Contents

1 Introduction

SINP bridges the gap between deterministic systems and probabilistic intent (NLP/LLMs). It formalizes the exchange of *Intent*, *Context*, and *Confidence* to allow a server to clarify, propose alternatives, or refuse requests deterministically before execution.

1.1 Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHOULD", and "OPTIONAL" are to be interpreted as described in RFC 2119.

2 Core Concepts and Formal Definitions

2.1 The Message Tuple

A message M in SINP is defined as a tuple:

$$M = (ID, CID, T, Sender, \Psi, \Gamma, \Phi, \Sigma) \tag{1}$$

Where:

- ID: UUIDv4 message identifier.
- CID: UUIDv4 conversation identifier.
- T: ISO-8601 UTC Timestamp.
- $Sender$: Identity object $\{id, auth\}$.
- Ψ: The Intent (Request) or Interpretation (Response) text.
- Γ: The Context object (Transcript/Summary).
- Φ: The scalar Confidence score, $\Phi \in [0, 1]$.
- Σ: Cryptographic signature (Optional).

2.2 Confidence (Φ)

Confidence is a scalar value representing *willingness to act*.

- Φ_c (Client): Certainty that Ψ reflects the user's desire.
- Φ_s (Server): Probability that the mapped capability c satisfies Ψ given Γ, adjusted for policy and safety.

3 Wire Format Specification

All messages MUST be serialized as JSON.

3.1 Request Schema

Note: The first message in a conversation MUST omit 'in_response_to'. All subsequent messages MUST include it.

```
{
  "protocol_version": "0.1",
  "message_id": "uuid-v4",
  "in_response_to": "uuid-v4 (OPTIONAL for INIT)",
  "conversation_id": "uuid-v4",
  "timestamp": "2025-12-31T12:00:00Z",
  "sender": { "id": "client_1", "auth_method": "token" },
  "intent": "string",
  "confidence": 0.85,  // REQUIRED, must be > 0
  "context": {
```

```
    "type": "transcript",
    "content": "...",
    "semantic_hash": "sha256_hex"
  },
  "constraints": {
    "max_cost": 10,
    "privacy": "private"
  },
  "signature": "base64_sig (OPTIONAL)"
}
```

Listing 1: Client Request Structure

3.2 Response Schema

```
{
  "message_id": "uuid-v4",
  "in_response_to": "uuid-v4",
  "conversation_id": "uuid-v4",
  "timestamp": "2025-12-31T12:00:01Z",
  "responder": {
      "id": "srv_1",
      "capabilities": ["cap_id_1:v1"]
  },
  "interpretation": {
    "text": "Mapped explicit action description",
    "confidence": 0.92
  },
  "action": "EXECUTE | CLARIFY | PROPOSE | REFUSE",
  "action_metadata": {
      // Contains 'result' if EXECUTE
      // Contains 'questions' if CLARIFY
      // Contains 'reason_code' if REFUSE
  },
  "alternatives": [  // OPTIONAL
    {
       "interpretation": "Alternative action description",
       "confidence": 0.80,
       "estimated_cost": 1.5,
       "capability_id": "cap_id_2:v1"
    }
  ],
  "confidence": 0.90
}
```

Listing 2: Server Response Structure

3.3 Capability Schema

Servers MUST define capabilities using stable identifiers. Clients MAY query these via a `DISCOVER` intent or out-of-band registry.

```
{
  "id": "fetch_profile:v1",
  "description": "Returns user profile data.",
  "inputs": ["user_id"],
  "privacy_level": "pii_sensitive",
  "cost_units": 1.0
}
```

4 Mathematical Model and Decision Logic

4.1 Interpretation Function

The server implements an interpretation function f:

$$f(\Psi_{req}, \Gamma) \to (\hat{\Psi}, c, \rho) \tag{2}$$

Where $\hat{\Psi}$ is the server's interpretation, c is the capability, and ρ is the raw model probability.

4.2 Confidence Derivation

Let $R(c)$ be the reliability factor of capability c, $A(res)$ be resource availability, and $P(pol) \in \{0, 1\}$ be the policy check.

$$\Phi_s = \min(1, \rho \cdot R(c) \cdot A(res)) \cdot P(pol) \tag{3}$$

4.3 Decision Boundary (δ)

The server determines Action A using thresholds τ_{exec} (default 0.85), $\tau_{clarify}$ (default 0.50), and τ_{accept} (default 0.50).

$$A = \delta(\Phi_s, \Phi_c) = \begin{cases} \textbf{EXECUTE} & \text{if } \Phi_s \geq \tau_{exec} \land \Phi_c \geq \tau_{accept} \\ \textbf{REFUSE} & \text{if } P(pol) = 0 \lor \text{Malformed} \\ \textbf{PROPOSE} & \text{if } \exists c' \neq c : \Phi_s(c') > \Phi_s(c) \\ \textbf{CLARIFY} & \text{if } \Phi_s < \tau_{exec} \end{cases} \tag{4}$$

5 State Machines

5.1 Server State Automaton

1. **RECEIVED**: Validate signature (JCS) and schema. Check Replay ($|T_{now} - T_{msg}| \leq 5s$).
2. **INTERPRETING**: Run $f(\Psi, \Gamma)$. Compute Φ_s.
3. **DECIDING**: Apply $\delta(\Phi_s, \Phi_c)$.
   - `EXECUTE` → Perform Action → State **DONE**.
   - `CLARIFY` → Generate Questions → State **NEGOTIATING**.
   - `PROPOSE` → Generate Alternatives → State **NEGOTIATING**.

5.2 Client State Automaton

- **INIT**: Send Request (no `in_response_to`).
- **REFINING**: Wait for Response.
  - On `CLARIFY`: User input → Update Context → Send Request.
  - On `PROPOSE`:
    * If Accepted: Send new Request referencing proposal ID.
    * If Rejected: Send new Request with modified intent.
  - On `EXECUTE`: Terminate (**SATISFIED**).

6 Interpretation Algorithms

6.1 Deterministic Scoring (Baseline)

For capability keywords K_c and intent vector V_{int}:

$$\text{Score}(c) = \frac{\sum_{k \in K_c} \mathbb{I}(k \in V_{int})}{|K_c|} \cdot w_{match} \tag{5}$$

6.2 LLM Calibration Observability

If using LLMs, raw confidence x MUST be calibrated (e.g., Platt Scaling). Servers SHOULD publish monitoring metrics, specifically the **Brier Score** (BS) to ensure claimed confidence matches empirical accuracy:

$$BS = \frac{1}{N} \sum_{t=1}^{N} (f_t - o_t)^2 \tag{6}$$

Where f_t is the forecast probability (Φ_s) and o_t is the outcome (1 if user accepted/successful, 0 otherwise).

7 Security Integrity

7.1 Semantic Hashing (Caching)

The semantic hash is used for caching interpretations. It MUST exclude timestamps to ensure identical intents hit the cache.

$$H_{sem} = \text{SHA256}(\text{normalize}(\Psi)||\text{normalize}(\Gamma)) \tag{7}$$

7.2 Replay Protection

Replay attacks are mitigated by checking the tuple (ID, CID, T). Servers MUST reject messages where $|T_{now} - T_{sender}| > 5000ms$ (configurable).

7.3 Signature Canonicalization

If `signature` is used, the JSON payload MUST be canonicalized using **JCS (RFC 8785)**.

1. Remove `signature` field.
2. Canonicalize remaining JSON.
3. Sign using sender's private key.
4. Append signature to original JSON.

8 Appendix: Refusal Codes

Common `action_metadata.reason_code` values:

- `malformed_context`: Semantic hash mismatch or invalid structure.
- `privacy_violation`: Request requires PII but privacy constraints forbid.
- `capability_missing`: No capability matches intent with $\Phi > 0.2$.
- `policy_violation`: Intent understood but forbidden by server rules.