Network Working Group
Internet-Draft
Intended status: Informational
Expires: 28 August 2026

R. Pioli
Independent

February 2026

Agent Registration and Discovery Protocol (ARDP)
draft-pioli-agent-discovery-01

Abstract

This document specifies the Agent Registration and Discovery Protocol (ARDP), a lightweight protocol for registering, discovering, and reaching autonomous software agents in distributed and federated environments. ARDP provides stable agent identities, dynamic endpoint resolution, capability advertisement (including protocol selection among MCP, A2A, HTTP, and gRPC), minimal presence signaling, and a security-first discovery control plane. ARDP is transport-agnostic and complementary to existing agent interaction protocols.

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 5 August 2026.

Copyright Notice

Table of Contents

1. Introduction

Autonomous and semi-autonomous software agents introduce challenges in discoverability, reachability, and interoperability. Agents may be ephemeral, mobile across execution environments, and implemented by heterogeneous vendors.

ARDP addresses stable addressing of agents whose runtime location changes, authorized discovery by identity and declared capabilities, capability-driven selection among interaction protocols (e.g., MCP, A2A, HTTP, gRPC), and minimal, privacy-aware presence signaling.

2. Design Goals

Stable Identity; Dynamic Reachability; Minimalism (control plane only); Security by Default; Federation-Friendly; Extensibility.

3. Non-Goals

ARDP is intentionally narrow in scope. The following are explicitly out of scope for this specification:

1. Agent-to-agent interaction, session management, task execution, and tool invocation protocols. These are addressed by interaction protocols such as MCP and A2A.

2. Identity governance frameworks, IAM policy languages, and organizational trust models. ARDP accepts identity attestations as inputs but does not define how they are issued or governed.

3. Runtime authorization token formats and enforcement proxies. ARDP provides discovery-time authorization scopes; runtime enforcement is delegated to interaction layers.

4. Post-execution evidence, audit trails, compliance logging formats, and non-repudiation mechanisms.

5. Billing, accounting, reputation, benchmarking, and other business frameworks.

6. Naming and bootstrap mechanisms beyond locating an ARDP authority. DNS-based discovery, well-known URIs, or other bootstrap mechanisms may be used alongside ARDP.

7. Centralization requirements. ARDP supports domain-scoped authorities and explicit federation without mandating a single global registry.

4. Terminology

The key words MUST, MUST NOT, REQUIRED, SHALL, SHALL NOT, SHOULD, SHOULD NOT, RECOMMENDED, NOT RECOMMENDED, MAY, and OPTIONAL are to be interpreted as described in RFC 2119 and RFC 8174.

* Agent: Autonomous software entity capable of initiating and receiving interactions.

* Agent Identifier (AID): Stable, namespaced identifier of the form agent:<local-id>@<authority>.

* Registrar: Service that accepts agent registrations and maintains bindings.

* Resolver: Service that resolves an AID to active endpoints.

* Endpoint: Network location and protocol tuple through which an agent can be reached.

* Capability: Declarative description of supported protocols and interaction modes.

5. Architecture Overview

ARDP defines a logical control plane composed of registrars and resolvers. Agents register their presence and capabilities with a registrar. Authorized clients query resolvers to obtain endpoint and capability information.

6. Layering and Composition

ARDP operates as a control-plane protocol for registration and discovery. It is designed to compose with other layers of the agent ecosystem:

* Transport Security: ARDP endpoints are typically secured using TLS, mTLS, or QUIC. Transport security is orthogonal to ARDP semantics.

* Identity Governance and Attestation: ARDP uses identity attestations as inputs for proof-of-control during registration and for authorization decisions. The issuance, revocation, and governance of these attestations are handled by external identity frameworks.

* Bootstrap Mechanisms: Clients may use DNS-based discovery, well-known URIs, or other mechanisms to locate ARDP registrars and resolvers. ARDP does not prescribe a specific bootstrap method.

* Interaction Protocols: Once an agent's endpoint and capabilities are discovered via ARDP, clients select an appropriate interaction protocol (MCP, A2A, HTTP, gRPC) based on the advertised capabilities. ARDP does not define interaction semantics.

ARDP does not specify runtime enforcement mechanisms, session semantics, or evidence/audit systems. These concerns are delegated to the appropriate layers.

7. Agent Identity

Each agent SHALL have a unique AID. The authority component denotes the administrative authority responsible for the identity.

Agents MUST prove control of an AID during registration using cryptographic credentials bound to that identity.

7.1. AID Syntax

An ARDP Agent Identifier (AID) MUST follow this grammar (ABNF per RFC 5234):

```
aid              = "agent:" local-id "@" authority
local-id         = 1*( ALPHA / DIGIT / "_" / "-" / "." / "/" )
authority        = dns-name / internal-name / opaque-authority
dns-name         = 1*( ALPHA / DIGIT / "-" / "." )  ; see IDNA2008 notes
internal-name    = 1*( ALPHA / DIGIT / "-" / "." )
opaque-authority = "tenant-" 1*( ALPHA / DIGIT / "-" )
```

Note: When internationalized domain names are used, implementers should follow the IDNA2008 RFC series.

7.2. Canonical Form

The canonical AID string is:

1. The literal prefix agent: in lowercase.

2. The authority normalized to lowercase (including IDNA2008 normalization where applicable); internal-name is lowercased as-is.

3. local-id is case-sensitive and MUST be preserved as sent.

4. AIDs MUST be compared using the canonical form.

8. Registration Model

8.1. REGISTER

A REGISTER request includes: AID; one or more endpoints; capability document (versioned); TTL; and cryptographic proof.

Registrations are soft-state and MUST be refreshed before expiration. Refresh is performed by sending a new REGISTER request with the same (aid, binding_id) pair.

8.2. DEREGISTER

An agent MAY explicitly remove its registration.

8.3. Registration Semantics

Registration is idempotent on (aid, binding_id). If a client sends the same (aid, binding_id), the server MUST treat it as a refresh.

If a client sends the same aid with a different binding_id, the server MUST return a conflict error unless the client has registry:override scope.

8.4. TTL and Refresh

The server MUST be authoritative for expires_at and SHOULD return it in responses.

Clients SHOULD refresh at <= 0.5 * ttl with random jitter.

The server MUST define and enforce TTL bounds and advertise them in a metadata resource (recommended: /.well-known/ardp/meta).

8.5. Meta Resource

This section defines a metadata resource for deployments using HTTPS bindings.

Path: GET /.well-known/ardp/meta

The response advertises server capabilities, TTL bounds, supported protocols, and proof-of-control requirements:

```
{
  "version": "1.0",
  "registrar_id": "ardp.example.com",
  "min_ttl": 30,
  "max_ttl": 3600,
  "default_ttl": 300,
  "supported_protocols": ["MCP", "A2A", "HTTP", "gRPC"],
  "supported_auth_methods": ["jws-proof-of-control"],
  "jws_required": true,
  "nonce_endpoint": "/.well-known/ardp/nonce",
  "supported_schema_versions": ["v0"],
  "compliance_mode": "standard"
}
```

When jws_required is true, the nonce_endpoint field indicates where clients acquire nonces for proof-of-control.

9. Discovery and Resolution

9.1. RESOLVE

RESOLVE maps an AID to active endpoints and capabilities. Access MUST be authorized via the registry:resolve scope.

9.2. QUERY

QUERY allows authorized discovery by capability or namespace. Results SHOULD be minimized to prevent metadata leakage.

Access MUST be authorized via the registry:query scope.

9.3. Privacy Defaults and Redaction

By default, QUERY returns only aid and status. Clients MAY request full details via a detail=full parameter.

If redaction applies, the server MUST omit restricted fields and include "redacted": true in the response.

10. Capabilities

Capability documents MAY include supported protocols (MCP, A2A, HTTP, gRPC), transport bindings, authentication mechanisms, modalities, rate or cost hints, and protocol-specific metadata. Capabilities are declarative and do not imply authorization.

10.1. Capability Bindings

Capabilities MUST include protocol-specific bindings when a protocol is declared.

11. Presence and Health

Presence is limited to: online, offline, degraded.

12. Security Considerations

Threats include identity spoofing, registration poisoning, unauthorized discovery, replay and downgrade attacks, and registrar compromise.

Mitigations include cryptographic identity proof, signed registrations, strict authorization, rate limiting, and audit logging.

12.1. Proof of Control

Proof-of-control is REQUIRED for REGISTER and refresh operations. RESOLVE and QUERY do not use proof-of-control; they are authorized via scopes (see Authorization Scopes below).

12.1.1. Signed Payload

Clients MUST present a JWS-signed proof when registering or refreshing. The signed payload consists of:

* The registration body, serialized as deterministic JSON
* A server-provided nonce
* An issued_at timestamp in RFC 3339 format

12.1.2. Deterministic JSON Serialization

The registration body MUST be serialized using deterministic JSON with the following rules:

* Object keys are sorted lexicographically (Unicode code point order)
* No insignificant whitespace between tokens
* Array element order is preserved

* Strings are encoded as UTF-8

* Numbers use minimal representation without trailing zeros

12.1.3. Nonce Acquisition

Clients acquire nonces via:

```
GET /.well-known/ardp/nonce
```

The response is a JSON object:

```
{
  "nonce": "abc123...",
  "expires_in": 300
}
```

Nonces are single-use or have a narrow replay window. The recommended TTL is 300 seconds. The nonce endpoint is advertised in the /meta response when JWS proof-of-control is required.

12.1.4. Verification

Servers MUST verify the JWS using a JWKS key set (RFC 7517) or a configured trust store.

Servers MUST enforce replay windows and clock skew tolerances.

12.1.5. Algorithm Support

Implementations MUST support ES256 (ECDSA using P-256 and SHA-256).

Implementations MAY additionally support: RS256, RS384, RS512, PS256, PS384, PS512, ES384, ES512.

EdDSA is not currently supported.

12.2. Authorization Scopes

Proof-of-control is used for REGISTER and refresh operations. Authorization scopes apply to all operations and are the primary access control mechanism for RESOLVE and QUERY.

Operations require the following scopes:

* registry:register - required for REGISTER

* registry:refresh - required for refresh (re-REGISTER with same binding_id)

* registry:resolve - required for RESOLVE

* registry:query - required for QUERY

* registry:deregister - required for DEREGISTER

* registry:override - allows registering with a different binding_id for an existing AID

13. Federation

Registrars MAY federate across domains via explicit trust relationships and policy agreements.

13.1. Federation Profile (Minimum)

Federation is allowed only between explicit trust anchors.

Responses from remote registrars MUST include provenance fields: origin_authority, origin_registrar_id, origin_signature.

Caches MUST honor remote TTL and mark records as federated.

14. Relationship to Existing Protocols

ARDP complements, and does not replace, agent interaction protocols such as MCP and A2A.

15. Wire Format Sketch (Non-Normative)

JSON over HTTPS is shown as an example binding. Alternative encodings (e.g., CBOR, gRPC) are possible.

15.1. Minimal HTTPS Binding

This section describes a minimal HTTPS binding for ARDP operations. All endpoints are under the /.well-known/ardp/ path prefix.

* GET /.well-known/ardp/meta - Returns server metadata including TTL bounds, supported protocols, and proof-of-control requirements.

* GET /.well-known/ardp/nonce - Returns a JSON object with nonce and expires_in fields for use in proof-of-control.

* POST /.well-known/ardp/register - Registers or refreshes an agent. Refresh is performed by re-registering with the same (aid, binding_id).

* POST /.well-known/ardp/deregister - Removes an agent registration.

* GET /.well-known/ardp/resolve?aid={aid} - Resolves an AID to endpoints and capabilities.

* GET /.well-known/ardp/query - Queries for agents by capability or namespace. Supports parameters: protocol, schema, limit, offset, detail. Default returns minimal data; use detail=full to request complete records.

15.2. Error Model

Servers MUST return errors with: code (stable error code), message (human-readable), and correlation_id (for tracing).

Required codes: invalid_aid, unauthorized, forbidden, conflict, not_found, expired.

16. IANA Considerations

16.1. Well-Known URI

IANA is requested to register the following URI suffix per RFC 8615 in the "Well-Known URIs" registry:

URI suffix: ardp

Change controller: IETF

Specification document(s): This document

16.2. Agent Identifier Namespace

The agent: prefix is used as an internal identifier namespace and is not registered as a URI scheme in this version.

17. Capability Schema v0

A minimal JSON schema for capability documents is provided as a companion artifact in the GitHub mirror.

18. Open Issues

* Capability schema evolution
* Privacy-preserving discovery
* Federation bootstrapping
* Consider adopting RFC 8785 (JSON Canonicalization Scheme) for deterministic JSON serialization in a future revision

19. References

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

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

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

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

[RFC5890] Klensin, J., "Internationalized Domain Names for Applications (IDNA): Definitions and Document Framework", RFC 5890, 2010, <https://www.rfc-editor.org/rfc/rfc5890>.

[RFC7515] Jones, M., Bradley, J., and N. Sakimura, "JSON Web Signature (JWS)", RFC 7515, 2015, <https://www.rfc-editor.org/rfc/rfc7515>.

[RFC7517] Jones, M., Bradley, J., and N. Sakimura, "JSON Web Key (JWK)", RFC 7517, 2015, <https://www.rfc-editor.org/rfc/rfc7517>.

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

Author’s Address

Roberto Pioli
Independent
Email: roberto.pioli@gmail.com