Differences Between Google A2A Protocol and IBM ACP Protocol

The Google Agent2Agent (A2A) protocol and IBM's Agent Communication Protocol (ACP) represent two distinct approaches to enabling AI agent interoperability, each with different architectures, philosophies, and use cases.

Core Architectural Differences

Communication Architecture

- Google's A2A protocol employs JSON-RPC 2.0 over HTTP/HTTPS as its communication foundation. It uses Server-Sent Events (SSE) for streaming and supports synchronous, asynchronous, and real-time communication patterns. The protocol is designed as a "web-native" solution with strong integration capabilities for cloud services and API gateways.
- IBM's ACP takes a RESTful HTTP-based approach. Unlike A2A's JSON-RPC requirement, ACP uses standard HTTP conventions that integrate seamlessly into production environments without requiring specialized communication methods. ACP supports multiple communication layers including RESTful HTTP.

Agent Discovery Mechanisms

- A2A utilizes an "Agent Card" system for discovery. These JSON documents serve as digital business cards containing agent identity, capabilities, service endpoints, authentication requirements, and interaction instructions. Agents can discover each other through well-known URIs (like `/.well-known/agent.json`), curated registries, or direct configuration.
- ACP emphasizes local-first discovery and autonomy. It supports offline discovery by embedding metadata directly into distribution packages, enabling discovery in secure, disconnected, or scale-to-zero environments. ACP agents can be discovered even when inactive, making it suitable for edge computing scenarios.

Philosophical Approaches

Design Philosophy

- A2A follows a **cross-platform interoperability approac**h, designed for enterprise-scale, multi-vendor environments. It emphasizes breaking down silos between different AI systems and enabling complex collaboration workflows across heterogeneous platforms. The protocol has backing from over 50 technology partners including major companies like Salesforce, Atlassian, and MongoDB.
- ACP adopts a **local-first**, edge autonomy philosophy. It prioritizes privacy-sensitive, bandwidth-constrained environments and focuses on local multi-agent orchestration. ACP is designed for scenarios where agents need to operate independently of cloud services or external registries.

Technical Features and Capabilities

Communication Patterns

- A2A supports rich state management including **sessions**, **tasks**, and **agent memory.** It provides comprehensive task delegation mechanisms with structured message envelopes carrying task IDs, metadata, and optional stream channels. The protocol supports multi-round collaboration, message/artifact flow, and user experience negotiation.
- ACP focuses on **lightweight**, **event-driven communication with low latency**. It supports both **synchronous** and **asynchronous** communication patterns, **streaming interactions**, and both **stateful** and **stateless** operation patterns. ACP uses MIME types for content identification, making it extensible to handle any data format without protocol modifications.

Security and Authentication

Security Models

A2A implements comprehensive security mechanisms including **OAuth2**, **API Key authorization,** and **capability scope limitation**. It's designed to integrate easily with enterprise-level security systems and supports standardized authentication schemes.
ACP emphasizes capability tokens and local security measures. Its local-first approach provides better control over data privacy and reduces dependency on external authentication services.

Use Cases

A2A :

- Large-scale distributed and cloud-native AI systems
- Cross-vendor agent environments requiring interoperability
- Enterprise workflows spanning multiple platforms and vendors
- Complex collaborative tasks requiring rich state management

ACP :

- Local multi-agent orchestration and development environments
- Edge computing, IoT, and robotics scenarios requiring low latency
- Privacy-sensitive environments where data sovereignty is crucial
- Scenarios with limited bandwidth or intermittent connectivity