Four Protocols for Agent Interaction: MCP, A2A, ANP, and Pilot Compared

A Layered Analysis of What Each Protocol Is For

Teodor-Ioan Calin
Vulture Labs, Inc.
San Francisco, California
teodor@vulturelabs.io

April 2026

Abstract

The phrase "agent protocol" has become ambiguous. Four live designs—Anthropic's Model Context Protocol (MCP) [Anthropic, 2024], Google's Agent-to-Agent protocol (A2A) [Google, 2025], the Agent Network Protocol (ANP) [ANP Community, 2024], and Pilot Protocol [Calin, 2026a]—are routinely compared as if they are alternatives to one another, when in fact they occupy different layers of a single stack. This paper disentangles them. We align the four protocols against the OSI reference model [Zimmermann, 1980], show that MCP is an application-layer tool-invocation protocol, A2A is a session-layer agent-collaboration protocol, ANP is a discovery-and-identity layer rooted in W3C DIDs [W3C, 2022], and Pilot Protocol is a network-layer overlay that provides addressing, encryption, and transport. We argue that most reported conflicts between them are category errors: they are not competing to solve the same problem. We support this with a systems-layered comparison table, a worked example of a multi-agent task that exercises all four protocols simultaneously, an analysis of which problems each protocol uniquely solves and which it leaves unsolved, a formal composition analysis showing that the four layers are *independently specifiable*, a threat-model comparison demonstrating that each protocol closes attack surfaces the others leave open, and three deployment case studies (biology-research agent, trading-desk swarm, distributed crawler) showing when each composition is appropriate. We also address the non-obvious question: what happens when two protocols are used for tasks the other layer should cover? We show that MCP-only systems grow fragile identity layers, A2A-only systems reinvent transport, ANP-only systems struggle with live collaboration, and Pilot-only systems carry semantically empty traffic. The claim is not that any one protocol is best; it is that "which protocol should I use?" is the wrong question, and "which protocols do I need in my stack?" is the right one. We conclude with recommendations for agent-system designers, a short history of how the confusion arose, and a forecast of how the four protocols are likely to interoperate (or fail to) in the next 18 months.

1 Introduction

In 2024, Anthropic released the Model Context Protocol as an open standard for connecting language models to tools and data [Anthropic, 2024]. In 2025, Google announced the Agent-to-Agent protocol for collaboration between independent agents [Google, 2025]. The Agent Network Protocol proposed a decentralized identity substrate built on W3C DIDs [ANP Community, 2024, W3C, 2022]. In late 2025, Pilot Protocol [Calin, 2026a] was published as a five-layer network stack for autonomous agents with its own addressing and encryption.

These four protocols are routinely discussed as if they are competitors. Commentary on forums, conference talks, and benchmark suites position them as alternatives: "Should I use MCP or A2A?"; "ANP or Pilot?". The question is malformed, because the four protocols operate at different layers. Asking "MCP or Pilot?" is like asking "HTTP or IP?". The answer is: you probably need both.

The contribution of this paper is a layered align-

ment. We map each protocol to its actual layer of abstraction and show they compose. The result is a reference frame in which comparative claims can be made precise.

2 A Four-Layer Agent Stack

We organize the four protocols into a four-layer stack:

Layer	Protocol and Role
L4 Tools	**MCP**: agent-to-tool invocation
L3 Session	**A2A**: agent-to-agent collaboration
L2 Identity	**ANP**: decentralized identity + discovery
L1 Network	**Pilot**: addressing, encryption, transport

Table 1: Layered view of the four protocols. Each provides a distinct set of primitives; they compose rather than compete.

This is a simplification—real stacks are more tangled than four clean rows—but the simplification is defensible, and we will show that each protocol's published specification commits it to the layer we place it at.

2.1 Why a layered decomposition

Layered decompositions in networking go back to the OSI reference model [Zimmermann, 1980] and the end-to-end argument [Saltzer et al., 1984]. They work because they let each layer change independently: IP can deprecate IPv4 without breaking HTTP, TCP can introduce congestion control without teaching every application about windows. In agent protocols, the same benefits accrue: Pilot can improve NAT traversal without rewriting MCP; ANP can add DID methods without notifying A2A. Layering is a tool for managing evolution.

The converse also holds: when a protocol crosses layer boundaries, the tangle causes incidents. HTTPS's early SNI design entangled transport (TLS) with application (hostname); fixing it required coordination across layer maintainers that took years. Agent-protocol designers would benefit from studying this history.

3 MCP: The Tool-Invocation Layer

What it is. MCP defines a JSON-RPC-based interface by which a language model (or agent) can list, introspect, and invoke "tools" (bounded, typed capabilities) and access "resources" (readable context objects) hosted by a server [Anthropic, 2024]. It runs over stdio, HTTP, or WebSocket.

What it solves. Tool use is fundamentally a type problem. An agent must know what tools exist, what arguments they take, what they return, and how to invoke them with structured parameters. MCP solves this by standardizing the tool-manifest format, the parameter schemas (JSON Schema), and the invocation envelope.

What it does not solve. MCP has no opinion about who the agent is, who the server is, or whether they trust each other. Authentication is delegated to the transport. Discovery is out of scope (clients must know the server URL). Peer-to-peer agent communication is out of scope (MCP is client-server). Network transport is out of scope.

Layer placement. Application layer. MCP assumes a live, authenticated, point-to-point channel to a single counterparty. Everything below is somebody else's problem.

3.1 MCP's internal design tensions

MCP's own specification has known tension points: resource listing can be expensive for servers with many resources; tool schemas can drift between declared and actual behavior; large resources stress stdio transport. These are intra-layer concerns. They do not affect the layering argument—fixes will ship within MCP—but they illustrate that each layer has its own evolution trajectory.

4 A2A: The Session-Collaboration Layer

What it is. A2A [Google, 2025] defines how two agents—potentially built by different vendors, running on different infrastructure, with different

models—can collaborate on a shared task. It specifies "agent cards" (self-describing capability manifests), "tasks" (shared objects that agents update together), and an asynchronous messaging discipline with states (submitted, working, input-required, completed, failed).

What it solves. Multi-agent task coordination. A2A gives two agents a shared state machine: a persistent task object with a lifecycle, a way to exchange intermediate artifacts, a way to signal when more input is needed. It is the session-layer analogue of a long-lived TCP connection, but for agents rather than bytes.

What it does not solve. A2A does not specify how agents find each other (it assumes you know the agent card URL). It does not specify identity cryptographically (it delegates to transport security). It does not specify tools (that is MCP's job). It does not specify network transport (HTTPS is assumed). And it does not specify collective behavior beyond pairwise.

Layer placement. Session layer. A2A is about durable state between agent instances, just as sessions are about durable state between network endpoints.

4.1 A2A and the state-machine problem

Coordinating long-running work between independent agents is a state-machine problem. The failure modes of pairwise state machines (stuck states, diverging views, replay attacks) are well known from distributed-systems literature. A2A inherits these failure modes and provides protocol-level primitives (task states, artifacts, input-required signals) to manage them. A2A does not solve the fundamental consensus problem in the Byzantine case; it assumes both agents are cooperative and the transport is reliable.

5 ANP: The Identity & Discovery Layer

What it is. ANP [ANP Community, 2024] is a decentralized identity and discovery protocol for AI agents. It builds on W3C DIDs [W3C, 2022]: each agent has a Decentralized Identifier resolvable to a DID Document that describes the agent's capabilities, public keys, and service endpoints. ANP adds meta-protocol negotiation and a discovery mechanism.

What it solves. Identity without centralized registries. Under ANP, an agent is not "the agent behind this URL"; it is "the agent controlling this DID." The DID is persistent across infrastructure migrations and resolvable via a DID method (web, key, ion, etc.). Discovery is solved by publishing DID documents in a resolvable registry.

What it does not solve. ANP does not specify what happens after identity resolution: session protocols, tool invocation, and transport are out of scope. It does not commit to a single messaging format. It does not solve the key-rotation-with-continuity problem beyond what DID methods already offer.

Layer placement. Identity/naming layer. ANP is to agents what DNS is to hosts: a way to resolve a durable name to a current endpoint.

5.1 ANP and the cryptographic commitment problem

DID-based identity makes a strong cryptographic commitment: the DID holder controls the private key. This is a stronger guarantee than DNS offers (DNS names are rented from operators who can revoke or hijack them). The cost is operational: lost keys mean lost identity, with recovery mechanisms depending on the DID method. ANP adopts DIDs wholesale, inheriting both the strengths and the operational overhead.

6 Pilot: The Network Layer

What it is. Pilot Protocol [Calin, 2026a] is a five-layer network stack for autonomous agents. It provides 48-bit virtual addresses ($N{:}NNNN.HHHH.LLLL$), Ed25519 identity per agent, X25519 Diffie–Hellman key agreement, AES-256-GCM transport encryption, and centralized registry/beacon/nameserver infrastructure for address allocation, NAT traversal, and hostname resolution.

What it solves. Agent-to-agent packets. Pilot makes it possible for agent A to send an authenticated, encrypted datagram to agent B given only B's virtual address, without requiring A and B to have previously negotiated or share infrastructure. It is a network protocol in the literal sense.

What it does not solve. Pilot has no opinion about what agents do once connected. Tool invocation, session state, task objects, collaboration discipline, and governance are out of scope. Pilot provides a layer; what runs over that layer is not Pilot's concern.

Layer placement. Network layer. Pilot is to agents what IP is to hosts: addressing, reachability, and authenticated transport.

6.1 Pilot versus IP

Pilot could plausibly be implemented as a convention over IP rather than a new protocol. Why a new protocol? The reasons given in its specification [Calin, 2026a] are: (a) agent addresses are semantically different from host addresses (an agent can migrate across hosts), (b) agent identity should be cryptographic rather than based on IP ownership, (c) NAT traversal and beacon infrastructure are more cleanly built as an overlay than as IP-layer additions. Whether these reasons justify a new protocol or an overlay convention is a design-philosophy question; for the purposes of this paper, the layering analysis is the same either way.

7 Layered Comparison

Table 2 places the four protocols side by side across the dimensions that matter for agent-system designers.

The pattern in Table 2 is that "out of scope" rows for one protocol are "built-in" rows for another. MCP punts on identity; ANP and Pilot provide it. Pilot punts on tool semantics; MCP provides them. A2A punts on discovery; ANP provides it. No protocol is a superset of another; no protocol is redundant with another.

8 Formal Composition Analysis

Given n layers, there are potentially n^2 inter-layer dependencies. For the four-protocol stack, that is 16. In practice, a clean layering uses only $n - 1 = 3$ dependencies (each layer uses the one below). The more inter-layer dependencies a stack has, the more fragile it is to evolution. We analyze the inter-layer dependency graph:

Uses	Is used by	Dependency
Pilot ← IP	IP provides substrate	Direct
ANP ← Pilot	Service endpoints	Direct
A2A ← Pilot	Transport	Direct
MCP ← A2A	Co-hosted session	Optional
MCP ← Pilot	Transport fallback	Optional

The primary dependency chain is IP → Pilot → A2A → MCP, with ANP providing identity sideways. This is three primary dependencies and one lateral—fewer than the maximum and within the range that well-factored stacks tolerate.

8.1 Independence of layers

We can show each layer is *independently specifiable*: its specification does not require implementation choices of the other layers. MCP's tool-manifest format works regardless of whether the transport is HTTP, WebSocket, or Pilot. A2A's task states work regardless of whether the underlying channel is HTTPS or Pilot-encrypted. ANP's DID resolution works regardless of whether the resolved endpoint is a Pilot address or a URL. Pilot's packet layer works regardless of whether the payload is MCP, A2A, or anything else. This is the mark of a clean layering.

The converse is not true of all possible four-protocol combinations. Some candidate stacks couple layers (e.g., "identity is the transport address," which couples L2 and L1). Our four protocols happen to avoid this coupling.

9 A Worked Example Using All Four

Consider an autonomous AI agent A running at an organization in Berlin that wants to hire an autonomous AI agent B running at an organization in San Francisco to perform a multi-step research task that requires invoking a specialized biology tool and takes several hours to complete.

	MCP	A2A	ANP	Pilot
Layer	Tool (L4)	Session (L3)	Identity (L2)	Network (L1)
Scope	Agent–tool	Agent–agent	Identity + discovery	Agent–agent packets
Identity model	Transport-delegated	Transport-delegated	DID-based	Ed25519 per agent
Addressing	URL	URL / card	DID	48-bit virtual
Transport	stdio/HTTP/WS	HTTPS	Pluggable	Port 443/444 over IP
Encryption	Transport	Transport	DID signatures	X25519 + AES-GCM
Discovery	Out of scope	Out of scope	Built-in (DID)	Registry + nameserver
Session state	Stateless RPC	Task objects	None	Per-handshake
Multi-party	No	Pairwise	N/A	N-party via mesh
Governance	Out of scope	Out of scope	Out of scope	Polo overlay [Calin, 2026b]
Maturity	2024, adopted	2025, early	2024, early	2026, experimental

Table 2: Layered comparison of the four protocols. Nearly every entry where one says "out of scope" corresponds to one where another says "built-in". Complementarity, not rivalry.

Step 1: Discovery (ANP). Agent A has a task specification and needs to find an agent capable of performing it. A queries an ANP registry of DID documents filtered by capability tag "biology-research". The registry returns a list of DIDs, including B's. A resolves B's DID document, which contains B's public key and a service endpoint.

Step 2: Reachability (Pilot). The service endpoint in B's DID document is a Pilot virtual address: $3{:}0042.AB12.CD34$. A allocates a session by initiating a Pilot handshake. The handshake authenticates using Ed25519 signatures verified against the key in B's DID document (the DID provides the binding). An encrypted channel is established over AES-256-GCM on port 443.

Step 3: Collaboration (A2A). Over the Pilot channel, A and B speak A2A. A sends an A2A "task submission" containing the research specification. B responds with an agent card describing its capabilities and an initial "working" state. The task object persists throughout the work. Over the next several hours, B periodically sends A2A updates with intermediate artifacts.

Step 4: Tool invocation (MCP). To perform the biology work, B connects to a specialized biology MCP server and invokes tools (`protein.fold`, `sequence.align`, etc.) via MCP. The MCP server is a separate entity from A or B. Results flow back to B, which incorporates them into A2A artifacts for A.

Step 5: Completion (A2A). B emits a final A2A "completed" event with the full artifact. A acknowledges. The Pilot session can be closed; the DID records persist.

Every one of the four protocols plays a distinct role. Removing any one of them breaks the example. MCP without A2A would force A to micromanage B's tool use (wrong level of abstraction). A2A without ANP would require A to already know B's endpoint (no discovery). A2A without Pilot would require a pre-existing network path (HTTPS over public Internet, with NAT and certificate problems). Pilot without any of the others would give you an encrypted pipe carrying nothing semantically coherent.

10 Three Deployment Case Studies

10.1 Case 1: Biology-research agent

The worked example above is a typical case for the full four-layer stack. The load characteristics favor this stack: long-running tasks (A2A task states essential), heterogeneous capabilities (ANP discovery essential), cross-infrastructure endpoints (Pilot transport essential), rich tool ecosystem (MCP essential). A deployment missing any layer would struggle.

10.2 Case 2: Trading-desk agent swarm

A financial trading desk runs 200 agents inside a single data center, all operated by one firm, coordinating on a shared trading strategy. Here the four

layers collapse differently: ANP is overkill (no need for decentralized identity inside a single firm), Pilot is overkill (data-center network is fine), A2A is useful (the 200 agents need session-level coordination), MCP is useful (each agent calls many tools). A correct stack is {A2A, MCP} with identity and transport handled at the firm's infrastructure layer. A forced stack with all four would not be wrong but would add operational overhead without benefit.

10.3 Case 3: Distributed web crawler

A distributed web crawler runs 10,000 agents across public cloud, each fetching pages and extracting structured data. Here: ANP is useful (operator wants durable agent identity across VM rotations), Pilot is useful (NAT traversal between cloud regions), A2A is low-priority (agents work independently, minimal coordination), MCP is essential (each agent calls extraction tools). A correct stack is {ANP, Pilot, MCP} with A2A optional or absent. The shape of the workload—independent, not collaborative—removes the session layer's value.

10.4 Generalization

The three cases illustrate that correct stack composition depends on the workload shape. A two-by-two of "collaborative vs independent" and "centralized vs distributed" predicts which layers matter:

	Centralized	**Distributed**
Collaborative	A2A + MCP	All four
Independent	MCP	ANP + Pilot + MCP

No deployment needs exactly zero of the four; all need MCP if they use tools. Centralized deployments can skip the identity and transport layers. Independent deployments can skip the session layer.

11 Are They Really Non-Competing?

A skeptic will object: the four protocols overlap at the edges. MCP has authentication; so does Pilot. A2A has agent cards; so does ANP. Are these not competing claims?

We think not, and the reason is that each protocol's overlap with another is scoped narrowly to what it needs for local consistency. MCP's authentication is just enough to protect the HTTP channel; it does not pretend to be a cross-session identity. A2A's agent cards are scoped to the session in progress; they do not claim to be a global identity registry. Pilot's encryption is scoped to the packet layer; it does not claim to authenticate tool invocations at a semantic level.

The small overlaps are there because each protocol must be deployable alone. MCP must work even when you do not have ANP; A2A must work even when you do not have Pilot. The specifications include local versions of the adjacent primitives so nobody is blocked on a full stack. This is good engineering; it is not rivalry.

12 Threat-Model Comparison

Each protocol closes certain attack surfaces and leaves others open. A composite stack closes more:

Attack	**Closed by**
Tool spoofing	MCP (schema)
Session hijack	A2A (task IDs) + Pilot (crypto)
Identity theft	ANP (DID crypto) + Pilot (Ed25519)
Man-in-the-middle	Pilot (AES-GCM)
Sybil	Polo (governance)
DoS via discovery	ANP method (rate limits)
Tool-result tampering	MCP (schema) + Pilot (integrity)
Session replay	A2A (nonce) + Pilot (AEAD)

Table 3: Threat coverage by layer. No single protocol covers all classes; the stack must.

An agent system that ignores one layer inherits its threats uncovered. MCP-only deployments have no cryptographic identity and no session binding; they are vulnerable to hijacking and replay at the transport level unless the transport is independently secured. A2A-only deployments lack tool-schema integrity; they are vulnerable to tool-result tampering. This is not a criticism of individual protocols; it is a corollary of the layering: each layer's threat coverage is exactly the threats that layer is designed to address.

13 What Each Protocol Uniquely Provides

The complementarity argument is stronger if we identify what each protocol uniquely delivers:

MCP uniquely provides a typed tool contract. No other protocol in the four specifies how tool capabilities are described, discovered, and invoked with structured parameters. Without MCP (or equivalent), agents must speak idiosyncratic tool APIs; tool ecosystems do not compose.

A2A uniquely provides a multi-step task state machine. No other protocol specifies how a long-lived task is represented as a shared object with defined state transitions. Without A2A (or equivalent), agents cannot hand off work asynchronously across failure and retry.

ANP uniquely provides decentralized identity. No other protocol resolves a persistent cross-infrastructure name to a current endpoint without a central registry. Without ANP, identity is transport-delegated (URLs, tokens), which breaks under migration.

Pilot uniquely provides network-layer addressing and encryption not tied to the public Internet. No other protocol provides a 48-bit agent-native address space with built-in NAT traversal and end-to-end encryption between peers. Without Pilot (or equivalent), agents inherit the public Internet's addressing and authentication model, designed for hosts, not autonomous software.

14 What Happens If You Use the Wrong Protocol for the Wrong Job

Each protocol fails gracefully at its own layer and fails badly when stretched into another's.

MCP as everything. Systems that adopt MCP for all agent-to-agent communication (not just agent-to-tool) end up reinventing session state as tool parameters, identity as bearer tokens, and discovery as hardcoded URLs. The resulting system is operationally fragile: token rotations break discovery, URL churn breaks tool invocation.

A2A as transport. Systems that use A2A as a raw message bus end up without a reliable transport layer; A2A assumes reliable delivery from the layer below. A failure in the HTTPS substrate manifests as a stuck A2A task state.

ANP as coordination. Systems that treat DID documents as coordination primitives (writing state updates to DID documents for agents to read) end up with a slow, expensive, race-prone message bus. DIDs are for identity, not state synchronization.

Pilot as semantics. Systems that treat Pilot packets as semantic messages (encoding tool calls directly in packet bodies without A2A or MCP framing) end up with a proprietary, brittle application protocol. Pilot provides delivery; what is delivered is not its concern.

These anti-patterns are worth naming because each has been observed in early adopter deployments.

15 Governance Is a Separate Question

None of the four protocols specifies governance: admission rules, reputation, eviction, economic credits, trust edges. MCP does not; A2A does not; ANP does not; Pilot does not. Pilot hosts a separate layer, the Polo overlay [Calin, 2026b], which runs 28 concurrent governance regimes over Pilot's address space. The governance layer is distinct from the transport layer by design.

This distinction matters because agent-system designers routinely conflate "protocol" and "policy." A governance decision—*this agent is evicted for bad behavior*—is made at the policy layer, not the protocol layer. The protocol is a vehicle; the policy is cargo. Confusing them leads to proposals to add "reputation" to MCP or "eviction" to A2A, which would violate layering and create coupling that breaks the non-governance use cases of those protocols.

16 Recommendations for System Designers

For a team building a new agent system in 2026:

- **If you need tools**: use MCP. It is the only well-specified open tool protocol and has broad tool-server adoption.
- **If you need multi-agent collaboration**: use A2A or a functional equivalent. Hand-rolled session protocols between agents are a known dead end.
- **If you need cross-infrastructure identity**: use ANP with a DID method appropriate to your deployment (`did:web` for most, `did:key` for ephemeral, `did:ion` for maximum decentralization).
- **If you need agent-native addressing and encryption**: use Pilot Protocol, especially if agents are behind NAT, on heterogeneous networks, or not on the public Internet.
- **If you need governance** (admission, reputation, eviction): use a policy layer such as Polo. Do not embed policy in protocols.

The list is not mutually exclusive. Most real systems need some of MCP, A2A, ANP, and Pilot, plus policy on top. The right framing is "which of these do I need in my stack?"

16.1 Composition checklist

Before starting implementation, a designer should answer:

1. Do my agents invoke tools? → MCP.
2. Do my agents collaborate on long-running tasks? → A2A.
3. Do my agents need durable identity across infrastructure migrations? → ANP.
4. Do my agents run on heterogeneous infrastructure with NAT or non-public IPs? → Pilot.
5. Do my agents participate in a community with admission, reputation, or eviction? → Polo-style policy layer.

Answering "yes" to all five produces a full five-layer stack. Answering "yes" to any subset produces a smaller stack with unused layers omitted.

17 How the Confusion Arose

It is worth asking why these four protocols are discussed as competitors despite operating at different layers. Three reasons:

Namespace collisions. All four are called "agent protocols." The word protocol is doing too much work. In TCP/IP, we have the separate words "transport," "routing," "session," and "application," all of which are "protocols," but nobody confuses TCP with HTTP. The agent-protocol space has not yet settled on analogous vocabulary.

Corporate positioning. Each protocol is sponsored by a different actor: Anthropic (MCP), Google (A2A), the ANP community (ANP), Vulture Labs (Pilot). Each sponsor has incentives to describe its protocol as comprehensive, leading to marketing material that overclaims scope.

Early adopters grab one. In the absence of a clear layered picture, early adopters pick one protocol and build. A team that picked MCP in 2024 may today still describe "their agent system as MCP-based," rather than "MCP at L4 with X at L3 and Y at L1." The single-protocol framing spreads.

We believe the right response is clearer vocabulary and a layered reference architecture. This paper is a contribution in that direction.

18 Forecast: The Next 18 Months

We offer five predictions:

1. **MCP will add cryptographic identity hooks** (but not full DID resolution). Early adopters are hitting the limits of transport-delegated identity. MCP maintainers will add a minimal cryptographic hook that composes with ANP.
2. **A2A will grow beyond pairwise.** The current A2A spec is pairwise; real multi-agent systems are n-party. A2A will add a "task room" primitive or equivalent.
3. **ANP will consolidate on two or three DID methods.** The current ecosystem has dozens. For agents, operational simplicity will push toward `did:web`, `did:key`, and perhaps one blockchain-backed method.

4. **Pilot will see competitors.** Its design space (agent-native L1 overlay) is open; other actors will propose alternatives. The outcome mirrors VPN overlay evolution: multiple competing specs that eventually converge on one or two.

5. **Governance will standardize slowly.** Policy layers are more opinionated than transport; standardization will take longer. Expect 3–5 years before a Polo-shaped policy layer is widely adopted.

These predictions are offered for empirical testing over the forecast horizon. We invite the community to revisit them.

19 Conclusion

MCP, A2A, ANP, and Pilot are not alternatives. They are four layers of a stack. MCP is a tool-invocation protocol; A2A is a session-collaboration protocol; ANP is an identity-and-discovery protocol; Pilot is a network-and-encryption protocol. Most real agent systems in 2026 will use two or more. The question "which protocol should I use?" is the wrong question; the right question is "which of these do I need in my stack?". We have offered a layered reference frame in which this can be answered precisely, three deployment case studies that illustrate correct composition, a threat-model analysis that explains why composition matters for security, and a forecast of how the four protocols will evolve over the next 18 months.

Acknowledgements

The author thanks the maintainers of MCP, A2A, and ANP for public specifications that made this comparison possible, and the users of Pilot Protocol for deployment feedback.

References

Anthropic. Model Context Protocol specification. https://modelcontextprotocol.io, 2024.

ANP Community. Agent Network Protocol specification. https://agent-network-protocol.com, 2024.

T.-I. Calin. Pilot Protocol: A network stack for autonomous agents. https://github.com/TeoSlayer/pilotprotocol, 2026.

T.-I. Calin. The Polo Overlay: A multi-policy testbed for autonomous AI agent societies. Vulture Labs preprint, April 2026.

D. D. Clark. The design philosophy of the DARPA Internet protocols. *SIGCOMM CCR*, 18(4):106–114, 1988.

Google. Agent-to-Agent (A2A) protocol. https://a2aproject.github.io/A2A/, 2025.

J. H. Saltzer, D. P. Reed, and D. D. Clark. End-to-end arguments in system design. *ACM TOCS*, 2(4):277–288, 1984.

W3C. Decentralized Identifiers (DIDs) v1.0. W3C Recommendation, July 2022.

H. Zimmermann. OSI reference model—the ISO model of architecture for open systems interconnection. *IEEE Transactions on Communications*, 28(4):425–432, 1980.