Agent Communication Protocols

A Technical Comparison of
A2A, MCP, ANP, and Pilot Protocol

Version 1.0 — March 2026

Teodor-Ioan Calin

Vulture Labs

https://vulturelabs.com

A systematic analysis of four protocols shaping agent-to-agent communication.

Abstract

The AI agent ecosystem in 2026 features four major communication protocols operating at different layers of the stack: Google's Agent-to-Agent (A2A) protocol for task orchestration, Anthropic's Model Context Protocol (MCP) for tool integration, the Agent Network Protocol (ANP) for decentralized agent networking, and Pilot Protocol for network-layer infrastructure. Each addresses a distinct problem—connecting models to tools, coordinating tasks between agents, establishing decentralized identity, or providing the underlying connectivity fabric—yet developers frequently conflate them as competing standards. This paper presents the first systematic technical comparison across seven dimensions: protocol layer, transport and encoding, identity and addressing, discovery, security and trust, NAT traversal, and scalability. We demonstrate that these protocols are complementary rather than competing, operating at different layers of what will become the full agent communication stack. We conclude with concrete composition patterns showing how they combine in practice.

Contents

1. Introduction

The year 2026 marks a turning point in how autonomous AI agents communicate. After decades of building software around human-operated APIs—REST endpoints designed for browsers, webhooks triggered by user actions, OAuth flows requiring human consent—the industry is converging on the realization that agents need their own communication primitives.

Four protocols have emerged to address this need:

- **MCP** (Model Context Protocol), created by Anthropic in November 2024 and donated to the AI Alliance and Linux Foundation in December 2025, standardizes how language models connect to external tools and data sources.
- **A2A** (Agent-to-Agent Protocol), created by Google in April 2025 and donated to the Linux Foundation in June 2025, defines how opaque agents delegate tasks to each other over HTTP.
- **ANP** (Agent Network Protocol), created by Gaowei Chang in 2024 and released under the MIT license, proposes a decentralized identity and meta-protocol negotiation layer for open internet agent communication.
- **Pilot Protocol**, created by Vulture Labs in 2025 and released under AGPL-3.0, provides a complete L3/L4 network stack—virtual addresses, ports, encrypted tunnels, NAT traversal—for agents.

These protocols are frequently discussed as if they are competing standards vying for the same niche. They are not. They operate at fundamentally different layers of the communication stack and solve fundamentally different problems. MCP connects a model to its tools. A2A orchestrates tasks between agents. ANP establishes decentralized identity and negotiation. Pilot Protocol provides the underlying network infrastructure that makes agents reachable in the first place.

This paper contributes the first systematic technical comparison of all four protocols. We analyze each across seven dimensions, identify their complementary nature, and present concrete composition patterns for using them together.

2. Protocol Overviews

2.1 MCP (Model Context Protocol)

The Model Context Protocol was announced by Anthropic in November 2024 as an open standard for connecting AI models to external tools, data sources, and computational resources [1]. In December 2025, Anthropic donated MCP to the AI Alliance and Linux Foundation, establishing independent governance. The current specification version is 2025-11-25.

MCP follows a host-client-server architecture inspired by the Language Server Protocol (LSP). A *host* application (such as an IDE or chat interface) contains one or more MCP *clients*, each maintaining a stateful session with an MCP *server*. Servers expose three categories of primitives:

- **Resources** — contextual data that can be attached to a model's context window (files, database records, live data feeds).
- **Tools** — executable functions that the model can invoke with structured parameters, producing structured results.
- **Prompts** — reusable prompt templates that guide model behavior for specific tasks.

Transport is provided via two mechanisms: `stdio` for local process communication, and `Streamable HTTP` (replacing the earlier HTTP+SSE transport) for remote connections. Authentication for remote servers uses OAuth 2.1 with PKCE, enabling secure third-party tool access without exposing credentials to the model.

MCP's design deliberately limits its scope to the model-tool boundary. It does not address agent-to-agent communication, discovery of remote agents, or network-level concerns like NAT traversal. Its strength lies in standardizing tool integration—a problem previously solved ad hoc by every framework independently.

2.2 A2A (Agent-to-Agent Protocol)

Google introduced the Agent-to-Agent Protocol in April 2025, targeting the interoperability gap between autonomous agents built on different frameworks [2]. The protocol was donated to the Linux Foundation in June 2025 and has attracted support from over 100 technology companies including AWS, Microsoft, Salesforce, SAP, ServiceNow, and Cisco.

A2A models communication between a *client agent* and a *remote agent*, where the remote agent is treated as opaque—its internal architecture, model choice, and tool usage are not exposed. This opacity is a deliberate design choice: agents are treated as services with capabilities, not as components to be inspected.

The protocol operates over HTTP using JSON-RPC 2.0 for structured messaging, with Server-Sent Events (SSE) for streaming responses and optional webhook callbacks for asynchronous notifications. Starting with version 0.3.0, gRPC transport is also supported. A v1.0 release candidate is in progress with significant structural improvements including unified message types and Agent Card signature verification.

Discovery centers on **Agent Cards**—JSON documents published at `/.well-known/agent-card.json` that describe an agent's capabilities, supported input/output modes, authentication requirements, and endpoint URL. Agent Cards serve a similar role to OpenAPI specifications but are designed specifically for agent-to-agent interaction.

A2A defines a task lifecycle state machine with states including `submitted`, `working`, `input-required`, `completed`, `failed`, and `canceled`. Tasks contain *messages* (communication turns) and *artifacts* (output objects such as files, images, or structured data). This lifecycle model enables long-running tasks with human-in-the-loop interaction.

Authentication is flexible, supporting API keys, OAuth 2.0, and mutual TLS. The protocol itself does not mandate a specific authentication mechanism, deferring to whatever the Agent Card declares.

2.3 ANP (Agent Network Protocol)

The Agent Network Protocol was created by Gaowei Chang (formerly of Alibaba) with the goal of enabling decentralized, open-internet communication between autonomous agents [3]. Released under the MIT license, ANP takes an identity-first approach inspired by W3C Decentralized Identifiers (DIDs).

ANP is organized into three layers:

1. **Identity Layer** — uses the W3C DID specification with a custom `did:wba` method ("Web-Based Agent"). Each agent's DID document is hosted at a well-known URL on its domain, containing public keys, service endpoints, and capability descriptions. This enables decentralized identity without requiring a blockchain.

2. **Meta-Protocol Layer** — defines a natural-language negotiation mechanism where agents can dynamically agree on communication formats and interaction patterns. Rather than mandating a fixed schema, ANP allows agents to negotiate protocols at runtime using their language understanding capabilities.
3. **Application Layer** — provides the Agent Description Protocol (ADP), a structured format for describing agent capabilities, similar in spirit to A2A's Agent Cards but built on JSON-LD and linked data principles.

Security in ANP relies on ECDHE (Elliptic Curve Diffie-Hellman Ephemeral) key exchange for end-to-end encryption and DID-based signatures for authentication. The trust model is rooted in domain ownership—an agent's DID is bound to its web domain, leveraging the existing DNS and TLS certificate infrastructure.

ANP has submitted an IETF Internet-Draft ("Framework for AI Agent Networks," draft-zyyhl-agent-networks-framework-01) proposing standardization of the agent networking framework [9]. An accompanying arXiv paper details the protocol's architecture and security model [4]. The project maintains an active open-source implementation and has conducted interoperability demonstrations with multiple agent frameworks.

2.4 Pilot Protocol

Pilot Protocol, created by Vulture Labs in 2025, takes a fundamentally different approach from the preceding protocols: rather than defining application-layer semantics for what agents say to each other, it provides the network infrastructure that makes agents reachable [5].

Pilot Protocol is a complete L3/L4 overlay network stack layered on top of IP/UDP. Its core primitives mirror those of the internet itself:

- **Addresses** — 48-bit virtual addresses in the format `N:NNNN.HHHH.LLLL`, partitioned into a 16-bit network ID and 32-bit node ID.
- **Ports** — 16-bit port numbers enabling multiplexed services on a single address (echo on port 7, DNS on 53, HTTP on 80, secure on 443, etc.).
- **Tunnels** — encrypted UDP tunnels between agents, carrying Pilot packets with a compact 34-byte binary header.
- **Transport** — a full transport layer with sliding window reliability, AIMD congestion control, flow control with advertised receive windows, Nagle algorithm, and automatic segmentation.

The security model is built on Ed25519 key pairs for identity and X25519 + AES-256-GCM for tunnel encryption. Trust is established through mutual handshakes: both agents must explicitly approve the connection before any data flows. Agents are **private by default**—they cannot be discovered, resolved, or contacted unless they have established trust. This inverts the default of HTTP-based protocols, where endpoints are publicly reachable unless protected.

A critical differentiator is built-in NAT traversal. Pilot Protocol implements STUN-based endpoint discovery, UDP hole-punching for restricted NATs, and relay through beacon servers for symmetric NATs. This means agents behind consumer routers, corporate firewalls, or cloud NAT gateways can communicate without requiring public IP addresses, reverse proxies, or VPN infrastructure. Given that approximately 88% of internet-connected devices sit behind NATs [6], this capability is essential for a truly distributed agent network.

The reference implementation is written in Go with zero external dependencies. It includes a daemon process, a registry/beacon rendezvous server, a CLI tool (`pilotctl`), and a gateway

for bridging Pilot and IP traffic.

3. Technical Comparison

3.1 Protocol Layer and Purpose

The most fundamental difference between these protocols is the layer at which they operate. Table 1 summarizes each protocol's position in the stack.

Table 1: Protocol layer and purpose

Protocol	**Layer**	**Purpose**	**Analogy**
MCP	Application (L7)	Model ↔ Tool	USB for AI
A2A	Application (L7)	Agent ↔ Agent tasks	HTTP for agents
ANP	Application (L7)	Agent identity + negotiation	DNS + TLS for agents
Pilot	Network (L3/L4)	Agent connectivity fabric	TCP/IP for agents

MCP operates at the boundary between a language model and its tools—it is fundamentally a tool integration protocol. A2A operates at the boundary between autonomous agents—it is a task orchestration protocol. ANP operates at the identity and negotiation layer—it is a decentralized identity and discovery protocol. Pilot Protocol operates below all three, providing the network connectivity that each assumes already exists.

This layering has a critical implication: these protocols are not competing for the same niche. They address orthogonal concerns. An agent can simultaneously use MCP to access local tools, A2A to delegate tasks to remote agents, and Pilot Protocol to establish the encrypted tunnels over which those A2A messages travel.

3.2 Transport and Encoding

Table 2: Transport and encoding comparison

	MCP	**A2A**	**ANP**	**Pilot**
Transport	stdio, HTTP	HTTP, gRPC	HTTPS	UDP
Encoding	JSON-RPC 2.0	JSON-RPC 2.0	JSON-LD	Binary (34B hdr)
Streaming	SSE	SSE + webhooks	—	Native (flow ctl)
Statefulness	Session-based	Task lifecycle	Stateless + JWT	Connection-based
Overhead	Medium	Medium	Medium	Low

MCP and A2A both use JSON-RPC 2.0 over HTTP, reflecting their application-layer orientation. This choice maximizes interoperability with existing web infrastructure but introduces per-request overhead: TCP handshakes, TLS negotiation, HTTP framing, and JSON parsing for every interaction.

ANP uses JSON-LD (JSON for Linked Data), enabling semantic interoperability through standardized vocabularies. This adds expressiveness at the cost of parsing complexity and payload size.

Pilot Protocol uses a compact binary encoding with a 34-byte packet header over raw UDP. This eliminates the overhead of HTTP framing and text-based encoding, resulting in significantly lower per-message costs—particularly important for high-frequency agent interactions such as streaming sensor data, real-time coordination, or pub/sub messaging.

3.3 Identity and Addressing

Table 3: Identity and addressing comparison

	MCP	**A2A**	**ANP**	**Pilot**
Identity	OAuth tokens	Agent Card URL	W3C DID	Ed25519 key pair
Addressing	URL / stdio	HTTP URL	DID URI	48-bit virtual addr
Persistence	Session-scoped	URL-lifetime	Domain-lifetime	Permanent
Portability	No	No	Domain-bound	Full

Identity models vary significantly. MCP delegates identity to OAuth—the model's identity is its access token, scoped to a session. A2A ties identity to an HTTP URL where an Agent Card is published; the agent *is* its endpoint. ANP uses W3C DIDs bound to web domains, decoupling identity from a specific server while still relying on DNS.

Pilot Protocol assigns each agent a permanent 48-bit virtual address derived from its Ed25519 key pair. This address persists across restarts, network changes, and physical migrations. An agent behind a home router today, a corporate firewall tomorrow, and a cloud VM next week retains the same address. No other protocol in this comparison offers this level of address permanence.

3.4 Discovery

Table 4: Discovery mechanisms

	MCP	**A2A**	**ANP**	**Pilot**
Primary	Manual config	Well-known URL	DID resolution	Registry lookup
Secondary	Community registry	Directory crawling	Search engines	Hostname + tags
Scope	Local	Public internet	Public internet	Trust-gated
Default	Configured	Discoverable	Discoverable	Private

Discovery approaches reflect each protocol's trust philosophy. MCP servers are manually configured by the host application—there is no built-in discovery mechanism, though community registries and DNS-SD have emerged as conventions.

A2A uses a well-known URL pattern (`/.well-known/agent-card.json`) that enables both targeted lookup and broad crawling. Any HTTP client can discover an A2A agent if it knows the domain. This openness facilitates ecosystem growth but also means agents are publicly enumerable by default.

ANP uses DID resolution, where an agent's DID document (hosted at a well-known path on its domain) contains service endpoints and capabilities. Like A2A, this is publicly discoverable via web crawling and search engine indexing.

Pilot Protocol takes the opposite approach: agents are **invisible by default**. The registry stores agent entries, but resolution is gated by trust—only agents that have completed a mutual handshake can resolve each other's addresses. Public agents can opt in to visibility, but the default is privacy. Agents can also register hostnames (human-readable names) and tags (capability descriptors) for discovery within their trust network.

3.5 Security and Trust

Security represents the most significant divergence between these protocols and merits detailed comparison.

Table 5: Security and trust comparison

	MCP	**A2A**	**ANP**	**Pilot**
Default visibility	Configured	Public	Public	Private
Authentication	OAuth 2.1 + PKCE	API key / OAuth / mTLS	DID signatures + JWT	Ed25519 signatures
Encryption	TLS (transport)	TLS (transport)	ECDHE E2E	X25519+AES-GCM E2E
Trust establishment	Client config	Fetch Agent Card	DID resolution	Mutual handshake
Revocation	Token expiry	HTTP 401/403	DID key rotation	Instant (untrust)
Enumeration	N/A	Crawlable	Crawlable	Blocked
Blast radius	Connected servers	Full network	Full network	Trust set only

Default visibility. MCP servers are not publicly addressable by design (they are local processes or configured endpoints). A2A and ANP agents are publicly discoverable through well-known URLs and DID resolution, respectively. Pilot agents are invisible: they cannot be discovered, resolved, pinged, or port-scanned unless they have established mutual trust with the querying agent.

Encryption. MCP and A2A rely on TLS at the transport layer—encryption terminates at the server, and any intermediary (load balancer, CDN, reverse proxy) sees plaintext. ANP provides end-to-end encryption via ECDHE key exchange. Pilot Protocol provides end-to-end encryption via X25519 Diffie-Hellman key agreement with AES-256-GCM authenticated encryption, with a random nonce prefix per connection to prevent replay attacks. In both ANP and Pilot, intermediaries (including the rendezvous server or relay) cannot read message contents.

Blast radius. If an A2A or ANP agent is compromised, the attacker can potentially interact with any agent on the network, since agents are publicly addressable. If a Pilot agent is compromised, the attacker can only reach agents in that agent's explicit trust set—other agents are invisible and unreachable. This containment property is a direct consequence of the private-by-default model.

Trust revocation. In Pilot Protocol, an agent can instantly untrust another agent, immediately severing connectivity. There is no token to wait for expiry, no cache to invalidate, no URL to return 403 from. The untrusted agent simply ceases to exist from the perspective of the agent that revoked trust.

3.6 NAT Traversal

NAT traversal is perhaps the most practically significant differentiator for real-world deployment. Approximately 88% of internet-connected devices operate behind Network Address Translation [6]—meaning they do not have publicly routable IP addresses and cannot receive unsolicited inbound connections.

Table 6: NAT traversal capabilities

	MCP	**A2A**	**ANP**	**Pilot**
NAT support	N/A (local) or reverse proxy	Requires public endpoint	HTTP-native, reverse proxy	Built-in
Full Cone	N/A	Requires config	Works	Direct
Restricted	N/A	VPN or tunnel	Works	Hole-punch
Symmetric	N/A	VPN or tunnel	Works	Relay
Setup required	None	Cloud infra	Web server	Zero

MCP's primary transport (stdio) is local and NAT is irrelevant. Its HTTP transport requires the server to be reachable, meaning a reverse proxy or tunnel service for agents behind NAT.

A2A assumes agents are reachable via HTTP URLs. An agent behind a NAT must provision a public endpoint—through a cloud deployment, a reverse proxy, a tunnel service like ngrok, or a VPN. This is a significant operational burden that limits A2A's applicability to agents running on developer laptops, IoT devices, or mobile platforms.

ANP, being HTTP-based, has the same fundamental limitation as A2A regarding inbound connections. However, because it uses standard HTTPS, it benefits from existing reverse proxy infrastructure and CDN support.

Pilot Protocol implements a three-tier NAT traversal strategy:

1. **Full Cone NAT** — STUN-based endpoint discovery. The agent's external address is valid for all peers; direct communication works immediately.
2. **Restricted / Port-Restricted Cone NAT** — UDP hole-punching coordinated by the beacon server. Both agents simultaneously send packets to each other's STUN-discovered endpoints, creating NAT bindings that allow bidirectional flow.
3. **Symmetric NAT** — relay through the beacon server. When hole-punching fails (because symmetric NATs assign different external ports per destination), traffic is relayed through the beacon with end-to-end encryption preserved.

This means any two Pilot agents can communicate regardless of their network topology, without any manual configuration, cloud infrastructure, or third-party tunnel services. The agent on a laptop behind a coffee shop WiFi can reach the agent on a Raspberry Pi behind a home router, which can reach the agent in a corporate data center—all using the same protocol, with the same security properties.

3.7 Scalability

The scalability characteristics of these protocols differ primarily in their connection overhead and coordination costs.

Connection overhead. HTTP-based protocols (MCP, A2A, ANP) incur per-request costs: TCP connection establishment (1 RTT), TLS handshake (1–2 RTTs), HTTP framing, and JSON serialization. While HTTP/2 and connection pooling mitigate some costs, each distinct agent pair still requires at least one TLS handshake. Pilot Protocol's UDP tunnels are established once and multiplex all communication between a pair of agents, amortizing setup costs across the lifetime of the relationship.

The N×(N-1)/2 problem. In a fully connected network of N agents, the number of pairwise connections grows quadratically. HTTP-based protocols face this challenge at the application layer, requiring each pair to establish and maintain separate HTTP connections. Pilot Protocol faces the same mathematical reality but addresses it at the network layer, where UDP tunnel multiplexing and the daemon's connection management reduce the per-connection overhead significantly.

Token tax. HTTP-based agent coordination carries what we term the "token tax"—the overhead of encoding structured data in JSON, wrapping it in HTTP, and parsing it back. For high-frequency interactions (hundreds of messages per second between cooperating agents), this overhead can reach 15× compared to binary protocols [5]. For infrequent task delegation (A2A's primary use case), this overhead is negligible.

Pub/Sub and multicast. Pilot Protocol provides built-in pub/sub messaging, enabling one-to-many communication patterns without establishing individual connections to each subscriber. MCP, A2A, and ANP do not provide native pub/sub; implementing broadcast patterns requires iterating over known peers and sending individual messages.

4. Complementarity

The central thesis of this paper is that MCP, A2A, ANP, and Pilot Protocol are not competing standards—they are complementary layers of what will become the complete agent communication stack. This section demonstrates concrete composition patterns.

4.1 The Layer Model

The four protocols map cleanly onto a layered architecture:

```
+-------------------------------------------------+
|  Application Services                           |
|  (MCP tools, A2A tasks, ANP negotiation)        |
+-------------------------------------------------+
|  Application Protocol Layer                     |
|  MCP (model-tool) | A2A (agent-task) | ANP (DID)  |
+-------------------------------------------------+
|  Network Layer                                  |
|  Pilot Protocol (addresses, ports, tunnels, trust) |
+-------------------------------------------------+
|  Physical Transport                             |
|  IP / UDP / Internet                            |
+-------------------------------------------------+
```

Just as HTTP (application layer) does not compete with TCP/IP (network layer), A2A does not compete with Pilot Protocol. They operate at different layers and combine naturally.

4.2 Pilot + MCP

An MCP server can be exposed over a Pilot tunnel, enabling remote access to tools without a public IP address. The MCP client connects to the server's Pilot address and port; the Pilot tunnel provides encryption, NAT traversal, and trust-gated access.

```
# Agent A runs an MCP server, exposed on Pilot port 80
# Agent B connects from across the internet
pilotctl connect <agent-a-address> 80
```

```
# MCP JSON-RPC flows over the encrypted Pilot tunnel
```

This composition provides several benefits over direct HTTP exposure: no public IP or reverse proxy required, end-to-end encryption (not just transport TLS), and trust-gated access (only trusted agents can reach the MCP server).

4.3 Pilot + A2A

A2A Agent Cards can advertise Pilot addresses instead of (or in addition to) HTTP URLs. Task requests and responses travel over Pilot tunnels, gaining NAT traversal and encryption automatically.

```
{
  "name": "research-agent",
  "pilot_address": "1:0001.0000.0042",
  "pilot_port": 80,
  "skills": [{"name": "web-research", "id": "web-search-v2"}],
  "authentication": {
    "type": "pilot-trust",
    "description": "Mutual Pilot handshake required"
  }
}
```

This composition replaces the assumption that agents must have HTTP URLs (and therefore public IP addresses) with the weaker assumption that agents have Pilot addresses (which work behind any NAT type).

4.4 Pilot + ANP

ANP's DID documents can reference Pilot endpoints as service entries, enabling DID-based identity verification over Pilot tunnels. The ANP identity layer provides decentralized, domain-anchored identity; the Pilot layer provides the connectivity.

```
{
  "@context": "https://www.w3.org/ns/did/v1",
  "id": "did:wba:example.com:agent:research",
  "service": [{
    "type": "PilotProtocol",
    "serviceEndpoint": "pilot://1:0001.0000.0042:80"
  }]
}
```

This composition combines ANP's strength (decentralized identity without blockchain) with Pilot's strength (connectivity without public endpoints).

5. Feature Matrix

Table 7 presents a comprehensive feature comparison across all dimensions discussed in this paper.

Table 7: Comprehensive feature matrix

Feature	**MCP**	**A2A**	**ANP**	**Pilot**
Layer	L7 (app)	L7 (app)	L7 (app)	L3/L4 (net)
Purpose	Model–tool	Agent tasks	Agent identity	Connectivity
Identity	OAuth token	Agent Card URL	W3C DID	Ed25519 key
Addressing	URL / stdio	HTTP URL	DID URI	48-bit virtual
Transport	stdio, HTTP	HTTP, gRPC	HTTPS	UDP
Encoding	JSON-RPC 2.0	JSON-RPC 2.0	JSON-LD	Binary (34B)
Encryption	TLS	TLS	ECDHE E2E	X25519+AES E2E
NAT traversal	N/A / proxy	Requires infra	Reverse proxy	Built-in
Trust model	Configured	Open by default	DID-anchored	Mutual handshake
Default visibility	Configured	Public	Public	Private
Discovery	Manual / registry	Well-known URL	DID resolution	Registry + tags
Pub/Sub	No	No	No	Built-in
Task delegation	No	Yes	Negotiated	Built-in
Tool calling	Yes	No	Negotiated	Via services
Streaming	SSE	SSE + webhooks	—	Native (flow ctl)
Offline/async	No	Polling	—	Inbox queuing
Dependencies	SDK	HTTP stack	HTTP + DID libs	Zero
License	Open	Open	MIT	AGPL-3.0
Governance	Linux Foundation	Linux Foundation	Community	Vulture Labs

6. Discussion

6.1 When to Use What

Based on our analysis, we propose the following decision framework:

Use MCP when a language model needs to call external tools—databases, APIs, file systems, code execution environments. MCP is the right choice for the model-tool boundary, particularly when tools are local or behind a controlled gateway.

Use A2A when autonomous agents from different vendors or frameworks need to delegate tasks to each other with structured lifecycle management. A2A is the right choice for agent interoperability at the application layer, especially in enterprise environments with existing HTTP infrastructure.

Use ANP when agents need decentralized, domain-anchored identity and the ability to negotiate communication protocols dynamically. ANP is the right choice when agent identity must be verifiable without a central authority and when interacting with agents across organizational boundaries on the open internet.

Use Pilot Protocol when agents need to communicate across networks, behind NATs, or without public IP addresses. Pilot is the right choice when the fundamental problem is connectivity—making agents reachable, keeping them private, and encrypting their communication end-to-end.

Use them together when building production agent systems. The most robust architecture uses Pilot Protocol for connectivity, A2A or ANP for agent-level interaction semantics, and

MCP for tool integration—each protocol at its appropriate layer.

6.2 The Layered Adoption Approach

We observe that protocol adoption follows a predictable pattern. Teams typically start with MCP (tool integration is the most immediate need), then adopt A2A (as they build multi-agent systems requiring interoperability), and finally encounter the connectivity problem that Pilot Protocol addresses (when agents need to communicate across networks without cloud infrastructure acting as intermediary).

This adoption path mirrors the history of internet protocols: applications were built first (email, file transfer), and the network layer (TCP/IP) was formalized once the need for universal connectivity became apparent. The agent ecosystem is undergoing the same transition—from ad hoc HTTP connectivity to purpose-built network infrastructure.

6.3 Open Questions

Several questions remain open as the agent protocol ecosystem matures:

- **Convergence.** Will the application-layer protocols (MCP, A2A, ANP) converge into a unified standard, or will they coexist as complementary specifications? The Linux Foundation's involvement in both MCP and A2A suggests coordination is possible.
- **Identity unification.** Can a single identity system serve all layers? Pilot's Ed25519 keys, ANP's DIDs, and A2A's Agent Card URLs represent three different identity models. A mapping or bridging mechanism would simplify multi-protocol deployments.
- **Performance.** As agent networks scale to millions of participants, the performance characteristics of HTTP-based vs. binary protocols will become increasingly significant. Empirical benchmarks at scale are needed.
- **Regulation.** As agents gain autonomy, regulatory frameworks may mandate specific security, identity, or auditability properties that favor certain protocol designs over others.

7. Conclusion

This paper has presented a systematic technical comparison of four protocols shaping agent-to-agent communication in 2026: MCP, A2A, ANP, and Pilot Protocol. Our analysis across seven dimensions—protocol layer, transport, identity, discovery, security, NAT traversal, and scalability—demonstrates that these protocols are complementary rather than competing.

MCP standardizes the model-tool boundary. A2A standardizes agent task orchestration. ANP provides decentralized identity and protocol negotiation. Pilot Protocol provides the network-layer infrastructure that makes agents reachable, private, and secure. Together, they form the layers of a complete agent communication stack.

The agent ecosystem does not need to choose one protocol. It needs all of them, each at its appropriate layer. The question is not "which protocol wins?" but "how do they compose?" This paper has shown that they compose naturally, with Pilot Protocol providing the connectivity fabric over which application-layer protocols operate.

As the number of autonomous agents grows from thousands to millions, the need for purpose-built network infrastructure—distinct from the human-oriented web—will only intensify. The protocols analyzed here represent the first generation of that infrastructure. Their complementary nature suggests the agent internet will, like the human internet before it, be built on layers.

References

[1] Anthropic. *Model Context Protocol Specification*. Version 2025-11-25, 2024–2025. https://spec.modelcontextprotocol.io/

[2] Google. *Agent-to-Agent (A2A) Protocol Specification*. 2025. https://google.github.io/A2A/

[3] G. Chang et al. *Agent Network Protocol (ANP) Specification*. 2024. https://github.com/agent-network-protocol/agent-network-protocol

[4] G. Chang, E. Lin, C. Yuan, R. Cai, B. Chen, X. Xie, Y. Zhang. *Agent Network Protocol Technical White Paper*. arXiv:2508.00007, July 2025.

[5] T.-I. Calin. *Pilot Protocol: A Network Stack for Autonomous Agents*. Version 1.8, Vulture Labs, 2026. https://github.com/TeoSlayer/pilotprotocol/blob/main/docs/WHITEPAPER.pdf

[6] J. Czyz et al. *Measuring IPv4 Address Sharing in the Wild*. ACM IMC, 2014.

[7] Anthropic. *Anthropic Donates MCP to Linux Foundation AI Alliance*. December 2025.

[8] Google. *A2A Joins the Linux Foundation*. June 2025.

[9] Y. Zhou, K. Yao, M. Yu, M. Han, C. Li. *Framework for AI Agent Networks*. IETF Internet-Draft draft-zyyhl-agent-networks-framework-01, October 2025.