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

L. Muscariello
M. Papalini
M. Sardara
S. Betts
Cisco

24 February 2026

An Overview of Messaging Systems and Their Applicability to Agentic AI
draft-mpsb-agntcy-messaging-01

Abstract

Agentic AI systems require messaging infrastructure that supports real-time collaboration, high-volume streaming, and dynamic group coordination across distributed networks. Traditional protocols like AMQP, MQTT, and NATS address some requirements but fall short on security, particularly regarding [AMQP] [MQTT] [NATS] post-compromise protection and quantum-safe encryption essential for autonomous agents handling sensitive data.

This document analyzes six messaging protocols—AMQP, MQTT, NATS, AMQP over WebSockets, Kafka, and AGNTCY SLIM—across dimensions critical for GenAI agent systems: streaming performance, delivery guarantees, security models, and operational complexity. We examine how each protocol's design decisions impact agentic AI deployments, from lightweight edge computing scenarios to large-scale multi-organizational collaborations.

AGNTCY SLIM emerges as a purpose-built solution, integrating Message Layer Security (MLS) [RFC9420] with gRPC [gRPC] over HTTP/2 [RFC7540] to provide quantum-safe end-to-end encryption, efficient streaming, and OAuth-based authentication [RFC6749]. Unlike transport-layer security approaches, SLIM's MLS implementation ensures secure communication even through untrusted intermediaries while supporting dynamic group membership changes essential for collaborative AI agents.

Discussion Venues

This note is to be removed before publishing as an RFC.

Source for this draft and an issue tracker can be found at https://github.com/agntcy/slim-spec.

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 28 August 2026.

Copyright Notice

Copyright (c) 2026 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (https://trustee.ietf.org/license-info) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document.

Table of Contents

1. Conventions and Definitions

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

2. Introduction

When designing a multi-agent system for generative AI, the messaging layer becomes a critical piece of infrastructure. GenAI agents—built with frameworks like LangGraph, AutoGen, or LlamaIndex—often need to collaborate in real time, exchange high volumes of streaming data (e.g., token-by-token outputs), and coordinate complex tasks such as voting or consensus. Moreover, security requirements extend well beyond basic TLS; in scenarios where agents share sensitive models or partial computations, post-compromise security and robust end-to-end encryption are essential.

In practice, you' ll want a protocol that efficiently handles one-to-many or many-to-many communication, supports dynamic membership (with agents joining or leaving on the fly), and scales to accommodate a "forest" of agents spread across global networks. Some protocols excel at ultra-low-latency, high-throughput streaming—critical for continuous token streams or aggregated embeddings—while others emphasize strong consistency and durability. Additionally, advanced cryptographic features such as automatic key rotation and forward secrecy are vital when compromised credentials must not enable an attacker to decrypt future communications.

Below, we compare six popular messaging protocols—AMQP, MQTT, NATS, AMQP over WebSockets, Kafka, and the emerging AGNTCY SLIM (Secure Low-Latency Interactive Messaging)—across dimensions that matter for GenAI agent systems: streaming performance, delivery guarantees, flexible pub/sub patterns, agent coordination, security (including end-to-end encryption and zero-trust support), and real-world adoption.

3. Protocol Analysis for Agentic AI Systems

The following sections provide detailed analysis of each messaging protocol in the context of agentic AI requirements.

3.1. Traditional Enterprise Messaging: AMQP

The Advanced Message Queuing Protocol (AMQP), most commonly implemented through RabbitMQ, represents the gold standard for enterprise messaging systems. AMQP's strength lies in its sophisticated message routing capabilities through exchanges, queues, and routing keys, enabling complex message flow patterns essential for enterprise applications.

For agentic AI systems, AMQP offers several advantages. Its support for both at-least-once and exactly-once delivery semantics (particularly in AMQP 1.0) ensures reliable message delivery between AI agents, which is crucial when agents are coordinating critical tasks or sharing expensive computational results. The protocol's durable queue support means that agent messages can persist across system restarts, preventing loss of important coordination data.

However, AMQP's enterprise focus comes with trade-offs. The protocol carries higher overhead due to its rich feature set, which may impact performance in high-frequency agent communication scenarios. Streaming capabilities require extensions like RabbitMQ Streams, adding complexity to deployments focused on real-time agent collaboration.

Authentication in AMQP relies on traditional enterprise mechanisms like SASL, LDAP, and Kerberos, which integrate well with existing corporate identity systems but may not align with modern cloud-native authentication patterns preferred in AI infrastructure.

3.2. IoT-Optimized Messaging: MQTT

Message Queuing Telemetry Transport (MQTT) emerged from the IoT world with a focus on lightweight, efficient communication over constrained networks. Its topic-based publish-subscribe model maps naturally to many agent communication patterns, where agents subscribe to topics representing different types of events or data streams.

MQTT's three Quality of Service levels (QoS 0, 1, and 2) provide flexibility in balancing performance versus reliability. For agentic AI systems, QoS 0 (at-most-once) works well for frequent status updates or non-critical notifications, while QoS 2 (exactly-once) ensures critical agent coordination messages are delivered reliably.

The protocol's very low overhead makes it attractive for scenarios involving large numbers of lightweight AI agents or edge computing deployments where bandwidth is constrained. However, MQTT's IoT heritage shows in its limitations for agentic AI use cases. Native streaming support requires broker extensions, and message-level security typically relies entirely on transport-layer TLS rather than end-to-end encryption.

MQTT's authentication mechanisms, while sufficient for IoT devices, may not provide the sophisticated identity and access management features required for complex multi-agent AI systems involving different trust domains.

3.3. Cloud-Native Messaging: NATS

NATS represents a modern approach to messaging designed for cloud-native architectures. Its lightweight design and support for multiple communication patterns—publish-subscribe, request-reply, and queue groups—make it particularly well-suited for microservices-based AI agent deployments.

The protocol's core at-most-once delivery semantics align well with scenarios where AI agents can tolerate occasional message loss in favor of high performance. For use cases requiring stronger guarantees, NATS JetStream provides at-least-once delivery and streaming capabilities, though this requires additional infrastructure complexity.

NATS's optional broker architecture offers interesting deployment flexibility for agentic AI systems. While most deployments use a broker for efficiency, the protocol can support peer-to-peer communication, potentially enabling direct agent-to-agent communication in specialized scenarios.

Authentication in NATS includes modern options like JWT tokens and NKey cryptographic authentication, aligning better with cloud-native security practices. However, like MQTT, NATS relies primarily on transport-layer security rather than providing built-in end-to-end message encryption.

3.4. Browser Integration: AMQP over WebSockets

AMQP over WebSockets addresses a specific deployment challenge: enabling browser-based AI agents or user interfaces to participate in AMQP-based agent coordination systems. This approach tunnels standard AMQP protocols through WebSocket connections, allowing web applications to overcome firewall restrictions and network topology limitations.

For agentic AI systems that include web-based components— such as user-facing AI assistants that need to coordinate with backend AI agents— this protocol variant provides a bridge between browser environments and enterprise messaging infrastructure. The WebSocket Secure (WSS) transport ensures encrypted communication from browser to broker.

However, the additional protocol layers (AMQP within WebSockets) introduce higher overhead compared to native AMQP or other lightweight protocols. This makes AMQP over WebSockets primarily suitable for scenarios where browser integration is essential rather than for high-performance agent-to-agent communication.

3.5. High-Throughput Streaming: Apache Kafka

Apache Kafka represents a fundamentally different approach to messaging, based on distributed commit logs rather than traditional message queues. This architecture provides exceptional throughput and built-in streaming capabilities that align well with certain agentic AI use cases.

Kafka’s partition-based topic model enables massive horizontal scaling, making it suitable for AI systems that need to process large volumes of training data, model updates, or inference results across distributed agent networks. The platform’s native streaming capabilities through Kafka Streams and KSQL provide powerful tools for real-time processing of agent-generated data.

The protocol’s built-in persistence across distributed clusters ensures that agent communication history is preserved and can be replayed, which is valuable for AI systems that need to audit agent decisions or retrain models based on historical interactions. Consumer groups enable multiple agents to process different partitions of the same topic concurrently, supporting parallel AI workloads.

However, Kafka’s strengths come with complexity costs. The requirement for a distributed cluster infrastructure may be overkill for simpler agent coordination tasks. While Kafka provides exactly-once semantics through transactions, the default at-least-once delivery may require additional deduplication logic in agent implementations.

Kafka’s security model, while comprehensive, relies primarily on transport-layer encryption and broker-based access controls rather than end-to-end message encryption, which may not meet the security requirements of AI systems handling sensitive model data or proprietary algorithms.

3.6. Next-Generation Agent Messaging: SLIM

AGNTCY SLIM (Secure Low-Latency Interactive Messaging) represents a purpose-built protocol for modern agentic AI systems, designed to address the specific security, performance, and coordination requirements that existing protocols cannot fully satisfy.

SLIM is intended as a transport layer for agent protocols like A2A and MCP. It handles secure routing, group messaging, and end-to-end encryption so protocol implementations can focus on agent semantics. A registration-based model lets agents become reachable through the SLIM network without exposing server ports, while only routing nodes need to be publicly reachable. This simplifies deployment for agents behind NATs and firewalls.

SLIM's foundation on gRPC over HTTP/2 and HTTP/3 provides several immediate advantages for AI agent communication. The binary protocol buffer wire format minimizes serialization overhead while supporting both binary and text data types essential for AI workloads. HTTP/2's multiplexing capabilities allow a single connection to carry multiple concurrent agent conversations, reducing connection overhead in systems with many interacting agents.

SLIM is architected as a distributed system with a clear separation of concerns:

* *Data plane*: Routes messages across SLIM nodes using only metadata for efficient forwarding and topology management.

* *Session layer*: Provides reliable delivery, MLS-based end-to-end encryption, and secure group management (create, invite, join, remove).

* *Control plane*: Orchestrates routing nodes, configuration, and administrative operations.

Routing nodes run only the data plane, keeping infrastructure lightweight, while language bindings include the data-plane client plus the session layer for full security and reliability.

The protocol's quality of service model explicitly addresses the diverse communication patterns found in agentic AI systems. Fire-and-forget messaging supports high-frequency status updates and non-critical notifications, while reliable exactly-once delivery ensures critical coordination messages and expensive computational results are never lost. This extends consistently across request-reply patterns and streaming communications.

Perhaps most significantly, SLIM's integration of Message Layer Security (MLS) provides quantum-safe, end-to-end encryption specifically designed for group communications. Unlike transport-layer security approaches used by other protocols, MLS ensures that messages remain secure even when transmitted through potentially compromised intermediaries—a critical requirement for AI systems operating across multiple trust domains.

The protocol's authentication model demonstrates particular innovation in addressing agentic AI security requirements. By transporting MLS credentials and cryptographic proofs within OAuth bearer tokens over HTTP/2, SLIM achieves several important properties:

* *Interoperability*: Leverages standard HTTP/2 and OAuth libraries, reducing implementation complexity and improving compatibility with existing infrastructure

* *Scalability*: Single persistent HTTP/2 connections efficiently carry many MLS-secured messages between agents

* *Immediate revocation*: Malicious or compromised agents can be immediately ejected by revoking their OAuth tokens without requiring complex ratchet tree rebalancing operations

SLIM's naming system is hierarchical and DID-inspired, for example: organization/namespace/service/instance. This supports anycast routing (to any available instance), unicast routing (to a specific instance), and service discovery without hardcoded endpoints or external registries. The structure maps cleanly to organizational boundaries and multi-tenant deployments.

The protocol's support for both broker-based and peer-to-peer operation offers deployment flexibility. While broker-based operation provides efficiency for multi-party group communications typical in agent coordination scenarios, peer-to-peer capabilities enable direct agent-to-agent communication when appropriate. SLIM exposes two session types that map to common agent patterns: point-to-point sessions for tool calls and group sessions for coordination and broadcast.

SLIM provides multi-language bindings. Python and Go bindings are available today, with JavaScript/TypeScript, C#, and Kotlin in progress, enabling heterogeneous agent systems to interoperate on the same transport.

3.7. Security Considerations for Agentic AI

Security requirements for agentic AI systems extend well beyond the capabilities provided by traditional messaging protocols. The autonomous nature of AI agents, combined with their access to sensitive data and computational resources, creates unique threat models that messaging infrastructure must address.

Post-Compromise Security: In traditional systems, credential compromise typically requires immediate revocation and re-authentication. However, AI agents may operate for extended periods with limited human oversight. SLIM's MLS implementation provides forward secrecy, ensuring that compromise of current credentials cannot decrypt past communications, and post-compromise security, guaranteeing that future communications remain secure even after credential compromise.

Quantum-Safe Cryptography: As quantum computing advances threaten current cryptographic standards, AI systems—which may operate for years with the same cryptographic keys—need protection against future quantum attacks. SLIM's quantum-safe MLS implementation provides this protection, while traditional protocols rely on classical cryptographic assumptions that may become vulnerable.

Multi-Domain Operations: Agentic AI systems often span multiple organizational and security domains, with agents from different organizations collaborating on shared tasks. Traditional protocols typically assume trust in messaging infrastructure, but SLIM's end-to-end encryption ensures secure communication even when messages transit through potentially untrusted intermediaries.

Dynamic Group Membership: AI agent groups frequently change as agents join collaborations, complete tasks, or become unavailable. MLS's efficient group key management handles these membership changes while maintaining security properties, unlike approaches that require complete cryptographic context regeneration.

3.8. Performance Implications

The performance characteristics of messaging protocols significantly impact the behavior and capabilities of agentic AI systems, particularly as the number of agents and frequency of interactions scale.

Latency Sensitivity: Many AI agent interactions are latency-sensitive, particularly in real-time decision-making scenarios or when agents are coordinating time-critical tasks. SLIM's HTTP/2 foundation provides header compression and multiplexing that reduce per-message overhead, while the binary protocol buffer encoding minimizes serialization costs.

Throughput Requirements: Large-scale agentic AI systems may involve thousands of agents generating substantial message volumes. While protocols like Kafka excel at raw throughput, they may introduce latency through their log-based architecture. SLIM balances throughput and latency through efficient connection reuse and optional reliability levels.

Connection Efficiency: Traditional protocols often require separate connections for each communication pattern or security context. SLIM's connection multiplexing allows a single HTTP/2 connection to handle diverse communication patterns between agents, reducing resource consumption and connection establishment overhead.

Streaming Performance: AI agents frequently exchange streaming data—such as token-by-token language model outputs or real-time sensor data. SLIM's native gRPC streaming support over HTTP/2 provides efficient bidirectional streaming without the overhead of connection-per-stream approaches.

3.9. Deployment and Operational Considerations

The operational characteristics of messaging protocols significantly impact the total cost of ownership and operational complexity of agentic AI systems.

Infrastructure Requirements: Traditional enterprise protocols like AMQP require dedicated message broker infrastructure with high availability and clustering capabilities. Kafka requires even more complex distributed infrastructure. SLIM's optional broker architecture allows deployments to scale infrastructure complexity with system requirements.

Monitoring and Observability: Debugging distributed agentic AI systems requires comprehensive visibility into agent communications. SLIM's foundation on standard HTTP/2 infrastructure enables use of existing observability tools and practices, while proprietary protocols may require specialized monitoring solutions.

Integration with Cloud Services: Modern AI deployments increasingly rely on cloud services for scalability and managed operations. SLIM's HTTP/2 foundation integrates naturally with cloud load

balancers, API gateways, and observability services, while specialized messaging protocols may require additional integration layers.

Compliance and Auditing: AI systems in regulated industries require comprehensive audit trails and compliance capabilities. SLIM's structured topic hierarchy and optional message persistence support regulatory requirements, while the end-to-end encryption provides compliance with data protection regulations.

4. RPC in Agentic Protocols and Relationship to Messaging

Most agent-facing interfaces in use today — notably A2A and the Model Context Protocol (MCP) — are Remote Procedure Call (RPC) oriented. They expose synchronous request/response semantics for tool invocation, resource listing, and capability execution. This section clarifies how RPC relates to asynchronous messaging and how the two paradigms interoperate in agentic systems.

4.1. RPC vs. Messaging: Synchronous vs. Asynchronous

* *RPC (A2A, MCP)*: Caller issues a request and blocks/awaits a timely response. Semantics emphasize tightly scoped operations (for example, "call tool X with parameters Y") with bounded latency and explicit error contracts.

* *Messaging (AMQP, MQTT, NATS, Kafka, SLIM)*: Decoupled producers/consumers communicate via topics/subjects/queues. Delivery can be one-to-one, one-to-many, or many-to-many with loose coupling, buffering, and retries. Producers are not inherently blocked by consumers.

In practice, agentic applications need both: synchronous tool invocations for interactivity and asynchronous channels for streaming output, progress, coordination, and fan-out/fan-in patterns.

4.2. When Asynchronous Feels Synchronous (Interactive Real-Time)

Asynchronous transports can provide an interactive, "RPC-like" experience when: - A request message includes a correlation ID and reply-to destination. - The callee publishes a response on the indicated reply destination within a short SLA. - Client libraries surface responses as futures/promises and manage timeouts/retries.

This underpins instant messaging UX and maps well to agent UIs where a user triggers an action and expects prompt, possibly streaming, results.

4.3. Bridging Patterns: RPC over Messaging and Streaming RPC

* *Request/Reply over Pub/Sub*: Implement RPC by publishing a command event and awaiting a correlated reply event (applies to AMQP, NATS, MQTT, and SLIM topics).

* *Streaming RPC*: Use bidirectional streams (for example, gRPC over SLIM HTTP/2/3) to deliver token streams, partial results, or progress updates while retaining an RPC caller experience.

* *Sagas and CQRS*: For multi-step workflows across agents, coordinate via asynchronous orchestration with idempotency keys, correlation/causation IDs, and compensations.

* *Backpressure and Flow Control*: Prefer streaming transports (HTTP/2/3, gRPC) or messaging systems with flow control when returning large/continuous results.

4.4. A2A and MCP in Context

* *A2A Agent Cards*: Describe capabilities/endpoints commonly invoked via RPC-style calls (tool execution, configuration). They benefit from messaging for discovery, eventing, and long-running workflows.

* *MCP*: Standardizes RPC-like interactions (resources, tools, prompts). For multi-party sessions, combine MCP RPC with a messaging layer for broadcast, presence, and coordination.

4.5. SLIM RPC (SRPC)

SLIM also supports a native RPC style via SRPC (Slim RPC), which layers request/response semantics on top of SLIM’s interactive, real-time messaging. SRPC addresses practical RPC concerns in distributed agent systems:

* Correlation and reply routing for synchronous calls over an async transport

* Idempotency keys and deduplication to make retries safe

* Lightweight synchronization/ordering guarantees for request/response and streaming

* Seamless fit for A2A/MCP-style tool calls while retaining SLIM’s MLS security and multiplexing

* Supports all four gRPC interaction patterns (unary, server streaming, client streaming, bidirectional streaming)

See SRPC reference and examples: SLIM RPC (SRPC) README (https://github.com/agntcy/slim/blob/main/data-plane/python/integrations/slimrpc/README.md).

4.6. Advantages of SLIM for A2A APIs

SLIM augments existing A2A-style RPC with capabilities that are difficult to achieve over plain request/response transports:

* *Simultaneous fan-out RPC (scatter-gather)*: Invoke a single RPC across many agents (by topic/group/labels) concurrently and aggregate responses (first-success, quorum, all-success) with correlation IDs.

* *Group addressing and dynamic membership*: Target MLS-secured groups; add/remove agents without reconfiguring endpoints.

* *Streaming responses*: Return partial results or token streams from each agent over a single multiplexed connection (gRPC over HTTP/2/3).

* *Idempotency and safe retries*: SRPC supports idempotency keys and deduplication, enabling robust retry without duplicating effects.

* *QoS, deadlines, and backpressure*: Apply delivery guarantees, per-call timeouts, and flow control to avoid overload while maintaining interactivity.

* *End-to-end security and multi-tenant isolation*: MLS E2E encryption and OAuth-based policy across both RPC and messaging channels.

* *Observability and tracing*: Correlation/causation IDs and standardized transport enable distributed tracing and per-agent metrics.

These capabilities let A2A-style tool calls scale beyond one-to-one interactions, enabling broadcast queries, coordinated multi-agent actions, and efficient collection of results in real time.

4.7. Security Implications

* *Identity and Authorization*: Reuse OAuth tokens across RPC (gRPC) and messaging (SLIM channels as topics) for consistent policy enforcement.

* *End-to-End Security*: MLS-backed secure channels (SLIM) so both RPC and messaging inherit end to end message encryption. ## Guidance: When to Choose What

* Use *RPC (A2A/MCP)* for low-latency, point operations with immediate feedback and well-defined error contracts.

* Use *Messaging* for broadcast/fan-out, decoupling, retries, buffering, and multi-party coordination.

* Use *Streaming RPC or RPC over Messaging* for interactive UX with partial/continuous results or uncertain duration operations.

5. Comparison

Table 1 provides a detailed comparison of three popular messaging protocols commonly considered for agent communication systems:

Feature	AMQP (e.g. RabbitMQ)	MQTT	NATS
Protocol Type	Message queueing (queues/ exchanges)	Lightweight pub/ sub for IoT	Lightweight messaging (pub/sub, req/ reply, queue groups)
Transport	TCP (optionally TLS)	TCP (optionally TLS)	TCP (optionally TLS)
Message Model	Queues, exchanges, routing keys	Topic-based	Subjects (pub/ sub), queue groups, request/reply
QoS / Delivery	At-least-once, exactly-once (AMQP 1.0)	QoS 0 (at-most-once), 1, 2 (exactly-once)	At-most-once (core), at-least-once with JetStream
Streaming	Via extensions/ plugins (e.g. RabbitMQ Streams)	Not native (requires broker extensions)	Native with JetStream

Persistence	Yes (durable queues)	Broker-dependent	Optional via JetStream
Protocol Overhead	Higher (rich feature set)	Very low	Very low
Broker Required	Yes	Yes	Optional (but common)
Authentication	User/password, SASL (e.g., LDAP, Kerberos)	Username/ password or custom tokens	NKey, JWT, token, user/ password
Transport Security	TLS	TLS	TLS
Message Security	Typically broker-level or plugin-based encryption	Usually none at message level; rely on TLS	None in core (TLS in transit), JetStream can encrypt at rest
Binary or Text	Binary framing	Binary framing	Text-based protocol (core), binary clients available
Use Cases	Enterprise messaging, financial transactions, RPC	IoT, mobile, sensor networks	Cloud-native microservices, real-time communications
Real-World Usage	Very widely used via RabbitMQ (top open-source broker) in enterprises of all sizes	Dominant in IoT ecosystems; supported by many device/ broker vendors	Gaining traction in cloud-native (CNCF project), used by major tech companies

Table 1

Table 2 extends the comparison to include additional protocols relevant to modern agentic AI systems:

Feature	AMQP over WebSockets	Kafka	SLIM
Protocol Type	AMQP tunneled through WebSockets	Distributed commit log, high-throughput pub/sub	Secure low-latency interactive messaging
Transport	WebSockets over TLS	TCP (optionally TLS)	gRPC (over HTTP/2-HTTP/3)
Message Model	Same as AMQP (depends on the broker's AMQP model)	Topics with partitions, consumer groups, offset-based consumption	Hierarchical names (org/namespace/service/instance), point-to-point and group sessions
QoS / Delivery	Same as AMQP	At-least-once default; exactly-once possible via transactions	Fire&Forget unreliable (at-most-once), unreliable and reliable (exactly-once). This extends to request/reply and streaming as well.
Streaming	Same as AMQP if broker supports streaming	Native log-based streaming (Kafka Streams, KSQL, etc.)	Native gRPC support via HTTP/2/3 client streaming, server streaming. Notice that Server Sent Events (SSE) with HTTP/1.1 cannot carry binary nor compressed data.
Persistence	Same as AMQP	Built-in: messages persist on disk across clusters	Not supported

Protocol Overhead	Higher (AMQP + WebSockets handshake)	Moderate (custom binary protocol, but optimized for high throughput)	Low: Wire format uses protocol buffer. Supports also binary (byte type in protobuf)
Broker Required	Yes	Yes (distributed cluster)	Yes for efficient multi-party. P2P is also possible.
Authentication	Same as AMQP (broker-based)	SASL/PLAIN, SASL/SCRAM, Kerberos, OAuth	Transports MLS credentials and proofs inside OAuth bearer tokens over HTTP/2. This gives you: Interoperability: Leverage standard HTTP/2 and OAuth libraries. Scalability: One persistent HTTP/2 connection carries many MLS messages. Immediate revocation: Eject bad actors by revoking their OAuth tokens—no need to rebalance the ratchet tree first.
Transport Security	WSS (WebSocket Secure)	TLS	TLS
Message Security	Same as AMQP (depends on the broker's encryption at rest/ in-transit)	TLS in-flight encryption, optional at-rest encryption (broker config)	MLS (Quantum safe, Secure end-to-end, even across insecure hops, post-compromise security)
Binary or Text	Binary	Binary	Binary or Text

	AMQP frames over WebSockets	protocol (common payloads: Avro, JSON, Protobuf)	
Use Cases	Browser-based apps needing AMQP behind firewalls	High-throughput data pipelines, streaming analytics, event sourcing	Group messaging, one-to-many, many-to-many, Cloud-native microservices, real-time communications, streaming
Real-World Usage	Less common, mainly for browser/ firewall scenarios using RabbitMQ or similar	Extremely widespread across industries; de facto standard for large-scale event streaming	New Entrant, low

Table 2

6. References

6.1. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <https://www.rfc-editor.org/rfc/rfc2119>.

[RFC6749] Hardt, D., Ed., "The OAuth 2.0 Authorization Framework", RFC 6749, DOI 10.17487/RFC6749, October 2012, <https://www.rfc-editor.org/rfc/rfc6749>.

[RFC7540] Belshe, M., Peon, R., and M. Thomson, Ed., "Hypertext Transfer Protocol Version 2 (HTTP/2)", RFC 7540, DOI 10.17487/RFC7540, May 2015, <https://www.rfc-editor.org/rfc/rfc7540>.

[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <https://www.rfc-editor.org/rfc/rfc8174>.

[RFC9420] Barnes, R., Beurdouche, B., Robert, R., Millican, J., Omara, E., and K. Cohn-Gordon, "The Messaging Layer Security (MLS) Protocol", RFC 9420, DOI 10.17487/RFC9420, July 2023, <https://www.rfc-editor.org/rfc/rfc9420>.

6.2. Informative References

[AMQP] OASIS, "OASIS Advanced Message Queuing Protocol (AMQP) 1.0 Specification", n.d., <https://www.oasis-open.org/standards#amqp>.

[gRPC] CNCF, "gRPC Documentation", n.d., <https://grpc.io/docs/>.

[Kafka] Foundation, A. S., "Apache Kafka Documentation", n.d., <https://kafka.apache.org/documentation/>.

[MQTT] OASIS, "OASIS MQTT Version 5.0 Specification", n.d., <https://docs.oasis-open.org/mqtt/mqtt/v5.0/mqtt-v5.0.html>.

[NATS] Communications, S., "NATS Documentation", n.d., <https://docs.nats.io/>.

[SLIM] AGNTCY, "AGNTCY SLIM Specification", n.d., <https://spec.slim.agntcy.org>.

Authors' Addresses

Luca Muscariello
Cisco
Email: lumuscar@cisco.com

Michele Papalini
Cisco
Email: micpapal@cisco.com

Mauro Sardara
Cisco
Email: msardara@cisco.com

Sam Betts
Cisco
Email: sambetts@cisco.com