SIPCORE Working Group
Internet-Draft
Intended status: Informational
Expires: 2 April 2026

T. McCarthy-Howe
VCONIC

29 September 2025

SIP Extension for Model Context Protocol (MCP)
draft-howe-sipcore-mcp-extension-00

Abstract

This document specifies a Session Initiation Protocol (SIP) extension to advertise support for, negotiate, and carry the Model Context Protocol (MCP). It defines: (1) a new SIP option-tag ("mcp"), (2) new header fields for capability advertisement and selection, (3) Contact feature-capability parameters for registration-time discovery, and (4) the "application/mcp+json" media type. MCP payloads can be exchanged during session establishment and mid-dialog using INVITE/200 (Offer/Answer), MESSAGE, and INFO.

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 2 April 2026.

Copyright Notice

Copyright (c) 2025 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. Code Components extracted from this document must include Revised BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Revised BSD License.

Table of Contents

1. Introduction

The Model Context Protocol (MCP) is an application protocol for structured interaction with tools and agents. While MCP enables powerful AI agent capabilities, real-world production deployments have revealed significant transport-layer limitations that impact reliability, performance, and user experience.

1.1. Problem Statement: MCP Transport Layer Failures

Current MCP implementations encounter measurable failures in production environments, particularly affecting latency, reliability, and scalability:

Performance Impact: Production deployments show MCP adds 300-800ms latency when invoked synchronously in critical transaction paths, with developers reporting this "destroys user experience" in customer-facing systems. P99 latency spikes cause substantial delays for the slowest 1% of transactions, leading to user frustration and cascading timeouts in orchestration flows.

Reliability Issues: Production scenarios report recovery failure rates of 20-30% without explicit error handling at the transport layer. STDIO pipes break silently, HTTP connection pools saturate under high load, and WebSocket connections disconnect-reconnect repeatedly, causing agents to lose context or fail mid-task.

Scalability Limitations: Connecting multiple tool servers (e.g., Github, Linear, Playwright) can consume over 60,000 tokens of context capacity, leading to expensive API overages and poor agent performance. Each MCP server operates in isolation with no shared state, forcing users to repeat steps or lose workflow progress between sessions.

Developer Experience: The official documentation for developing custom transports is lacking, the concepts section is complex, and the Python SDK lacks foundational interfaces, creating significant barriers to adoption and reliable implementation.

1.1.1. Summary of MCP Transport Pain Points

Failure Mode/Metric	Current MCP Impact	Real-World Evidence
High Latency (300-800ms)	Synchronous MCP flows	"Destroys user experience"
Connection Instability	STDIO pipes, WebSocket	"Pipes break silently"
Context/Token Bloat	Multiple tool servers	"60,000 tokens consumed"
Isolation, No State	Multi-step workflows	"Users repeat steps"
Lack of Documentation	Custom transport dev	"Documentation lacking"
P99 Latency Spikes	Tail latency in flows	Cascading timeouts

Table 1

1.2. SIP as a Solution

SIP is widely deployed for rendezvous, session negotiation, and inter-domain federation. This document defines a minimal, backward-compatible SIP extension enabling MCP-aware endpoints to discover each other and exchange MCP messages using existing SIP methods, addressing the transport-layer limitations identified in current MCP deployments.

1.3. Use Cases Addressed

This SIP extension for MCP addresses both general AI agent communication needs and specific scenarios that are uniquely enabled by SIP’s architectural capabilities.

1.3.1. General MCP Use Cases Enhanced by SIP

Enterprise AI Agent Orchestration: Organizations deploying multiple specialized AI agents (document processing, customer service, data analysis) require reliable, low-latency communication between agents. SIP's session management eliminates the 300-800ms latency penalties documented in current HTTP-based MCP deployments, while its proxy infrastructure enables intelligent routing based on agent capabilities.

Multi-Modal AI Interactions: Modern AI applications increasingly combine text, voice, and visual processing. SIP's media negotiation framework allows simultaneous audio streams (for voice interaction) and MCP data exchange (for tool calls and structured responses), enabling natural voice-guided AI workflows that are impractical with current MCP transports.

Cross-Organizational AI Collaboration: AI agents from different organizations need to collaborate while respecting security boundaries and policies. SIP's mature inter-domain federation model provides the trust management and policy enforcement mechanisms necessary for secure cross-organizational agent interactions.

High-Availability AI Services: Production AI systems require robust failover and load distribution. SIP's registration-based discovery provides 60-120 second agent availability updates (vs. 5-10 minutes with DNS), while proxy-based load balancing eliminates the single points of failure common in current MCP deployments.

1.3.2. SIP-Unique Use Cases

Voice-First AI Agent Interactions: Call centers, voice assistants, and telephony-integrated AI systems require tight coordination between voice streams and AI tool execution. SIP's native audio handling combined with MCP tool calls enables scenarios like: - Customer service agents that can simultaneously talk to customers and execute backend tool calls - Voice-controlled document processing where spoken commands trigger complex AI workflows - Real-time language translation with tool-assisted context lookup

Telecommunications-Integrated AI: Existing SIP infrastructure in telecommunications and enterprise environments can be extended to support AI agents without requiring parallel communication systems: - PBX systems can route calls to AI agents based on detected capabilities - Existing SIP monitoring and billing systems can track AI agent usage - Telecom-grade reliability and security models apply to AI agent communications

Session-Aware AI Workflows: Long-running AI processes that maintain conversational context across multiple interactions benefit from SIP's dialog management: - Multi-step document review processes where agents maintain state across sessions - Collaborative AI workflows where multiple agents contribute to extended tasks - Educational AI tutors that maintain learning context across multiple sessions

Multimedia AI Tool Calling: The combination of MCP with MSRP enables sophisticated multimedia AI interactions: - Image analysis agents that receive binary image data without base64 encoding overhead - Document processing agents that can stream large generated reports in real-time - Creative AI agents that exchange multimedia assets (images, audio, video) as part of tool calls

1.3.3. Performance-Critical Use Cases

Real-Time AI Decision Making: Applications requiring sub-second AI responses benefit from SIP's persistent session model: - Financial trading systems with AI-assisted decision making - Industrial control systems with AI-based optimization - Emergency response systems with AI-powered resource allocation

High-Throughput AI Processing: Batch processing scenarios where multiple AI agents need to coordinate efficiently: - Large-scale document processing pipelines - Distributed AI training coordination - Parallel data analysis workflows

1.3.4. Regulatory and Compliance Use Cases

Auditable AI Interactions: Industries with strict audit requirements can leverage SIP's mature logging and monitoring ecosystem: - Healthcare AI systems requiring HIPAA compliance - Financial AI systems requiring transaction audit trails - Government AI systems requiring security clearance-based access control

Privacy-Preserving AI Federation: Organizations needing to collaborate while maintaining data sovereignty: - Healthcare research collaborations across institutions - Financial consortium AI without data sharing - Government intelligence sharing with compartmentalized access

1.3.5. Migration and Integration Use Cases

Gradual MCP Transport Migration: Organizations can incrementally adopt SIP-based MCP without disrupting existing systems: - Hybrid deployments supporting both HTTP and SIP transports - Phased migration from WebSocket to SIP-based agent communication - A/B testing of transport performance in production environments

Legacy System Integration: Existing SIP infrastructure can be extended to support modern AI capabilities: - Contact centers adding AI agents to existing SIP-based phone systems - Enterprise communications platforms integrating AI assistants - Telecommunications providers offering AI services through existing SIP infrastructure

These use cases demonstrate that while SIP adds implementation complexity compared to simpler transports like HTTP, it enables entirely new classes of AI agent interactions that are impractical or impossible with current MCP transport mechanisms. The extension is particularly valuable for organizations with existing SIP infrastructure, real-time performance requirements, or complex inter-organizational collaboration needs.

1.4. Architectural Justification

1.4.1. Why SIP for MCP Transport?

While MCP can operate over various transports including HTTP and WebSocket, SIP provides unique architectural advantages that make it particularly suitable for agent-to-agent communication scenarios:

Session Management and State: SIP's inherent session model aligns naturally with MCP's stateful conversation paradigm. Unlike stateless HTTP interactions, SIP dialogs provide persistent session context that can maintain MCP conversation state, tool availability, and capability negotiations throughout the interaction lifecycle.

Rendezvous and Discovery: SIP's registration and location services enable dynamic discovery of MCP-capable agents across network boundaries with superior performance characteristics compared to DNS-based alternatives. SIP registrations provide programmable TTLs (60-3600+ seconds) with immediate effect, enabling rapid agent deployment and failover scenarios that are impractical with DNS propagation delays (typically 300+ seconds).

Inter-domain Federation: SIP's mature federation model allows MCP interactions to span organizational boundaries securely. This enables scenarios where agents from different organizations can collaborate while respecting domain policies and security boundaries.

1.4.2. Limitations of HTTP/WebSocket-Only Approaches

Real-world MCP deployments have demonstrated concrete failure modes and performance limitations with current transport approaches:

HTTP Transport Failures: - Lacks built-in session management requiring application-layer session tracking - No standardized discovery mechanism for dynamic agent location; DNS-based discovery suffers from propagation delays (300+ seconds) making rapid deployment and failover impractical - Limited support for inter-domain routing and federation - Requires additional infrastructure for load balancing and failover - *Production Impact*: HTTP connection pools saturate under high load, causing timeouts that are difficult to correlate with specific upstream errors - *"Universal Router Trap"*: Teams routing every user query through MCP over HTTP add hundreds of milliseconds to critical flows (e.g., e-commerce checkout), leading to lost conversions and board-level escalation of failures

WebSocket Transport Failures: - Requires pre-established HTTP connection setup - No inherent support for multi-party sessions or session transfer - Limited routing capabilities for complex network topologies - Lacks standardized capability advertisement mechanisms - *Production Impact*: Persistent connections disconnect and reconnect repeatedly under real-world network variability, causing agents to lose context or fail mid-task - *Reliability Issues*: Error rates above 0.1% indicate systemic issues, with recovery failure rates of 20-30% without explicit error handling

STDIO Transport Failures: - *Silent Failures*: STDIO pipes break silently, leading to mysterious dropped connections that are not detected until a downstream process fails - *Process Management*: Difficult to monitor and manage process lifecycle in production environments - *Scalability*: Limited to single-process communication patterns

1.4.3. SIP’s Value-Added Capabilities

Advanced Routing: SIP’s proxy infrastructure enables sophisticated routing based on MCP capabilities, load distribution, and policy enforcement. Proxies can inspect MCP-Capabilities headers to route requests to appropriate agents.

Session Mobility: SIP’s re-INVITE mechanism allows MCP sessions to be transferred between agents or modified mid-conversation, enabling scenarios like agent handoff or capability escalation.

Multi-modal Integration: SIP’s media negotiation framework allows MCP data exchange to be combined with audio/video streams, enabling rich multi-modal agent interactions (voice + tool calls).

Security and Privacy: SIP's established security model (TLS, S/MIME, SIPS) provides end-to-end security for sensitive MCP interactions, with well-understood privacy and authentication mechanisms.

1.4.4. Discovery Performance Analysis: SIP vs. DNS

A critical architectural advantage of SIP-based MCP transport lies in its superior discovery performance characteristics:

DNS-Based Discovery Limitations: - Standard DNS TTL values (300-3600 seconds) create significant delays for agent availability updates - DNS cache invalidation requires waiting for TTL expiration across all resolvers in the path - Reducing TTLs below 60 seconds increases authoritative server load and is often impractical - Global DNS propagation can take 5-15 minutes for cross-domain scenarios - DNS is optimized for relatively static records, not dynamic service availability

SIP Registration Performance Advantages: - Registration refresh intervals programmable from 60 seconds to hours based on agent characteristics - Immediate effect upon registrar receipt - no propagation delays - Failed registrations detected within one refresh interval (60-180 seconds typical) - Explicit de-registration provides immediate service removal - Bulk capability updates possible in single REGISTER transaction - Local consistency within registration domain eliminates cache coherency issues

Quantitative Performance Comparison: - Agent deployment: 60-120 seconds (SIP) vs. 5-10 minutes (DNS) - Failover detection: 60-180 seconds (SIP) vs. 5-15 minutes (DNS) - Cross-domain discovery: 60-300 seconds (SIP peering) vs. 5-15 minutes (global DNS) - Capability updates: Immediate (SIP) vs. TTL-dependent (DNS)

This performance differential is critical for AI agent ecosystems requiring rapid adaptation to changing agent availability and capabilities.

1.4.5. Quantitative Performance Analysis: SIP vs. Current MCP Transports

Real-world deployment data demonstrates significant performance advantages of SIP-based transport over current MCP approaches:

Latency Comparison: - *Current MCP over HTTP*: 300-800ms added latency in production systems - *SIP-based MCP*: Sub-100ms for signaling, with persistent session context eliminating repeated handshakes - *P99 Latency*: SIP’s session-oriented model reduces tail latency by maintaining persistent connections vs. HTTP’s per-request overhead

Reliability Metrics: - *Current MCP Transports*: 20-30% recovery failure rates without explicit error handling - *SIP-based MCP*: Built-in error handling and recovery mechanisms with standardized error codes - *Connection Stability*: SIP’s dialog management provides explicit session state vs. silent failures in STDIO/ WebSocket

Scalability Characteristics: - *Current MCP*: 60,000+ tokens consumed by multiple tool servers, causing API cost overages - *SIP-based MCP*: Capability negotiation and filtering reduces unnecessary context transmission - *Session Management*: Persistent SIP dialogs maintain state vs. stateless HTTP requiring repeated context establishment

Developer Experience Improvements: - *Current MCP*: Lacking documentation and complex custom transport development - *SIP-based MCP*: Leverages mature SIP ecosystem with extensive tooling, libraries, and operational experience - *Standardization*: Well-defined extension points vs. ad-hoc transport implementations

1.4.6. Specific Use Cases Addressing Real-World MCP Problems

Avoiding the "Universal Router Trap": Organizations currently experiencing 300-800ms latency penalties from routing every user query through MCP can use SIP’s capability-based routing to selectively engage MCP only when needed, with proxies routing based on MCP-Capabilities headers to appropriate specialized agents.

Enterprise Agent Federation with Shared State: Large organizations struggling with isolated MCP servers that force users to repeat steps can leverage SIP’s session management to maintain persistent agent context across departmental boundaries, with secure, policy-controlled inter-agent communication through SIP’s domain-based routing.

High-Availability Agent Deployments: Production environments experiencing 20-30% recovery failure rates can benefit from SIP’s built-in error handling, automatic failover mechanisms, and proxy-based load distribution, eliminating silent failures common in STDIO pipes and WebSocket disconnections.

Cross-Vendor Agent Interoperability: Organizations facing integration complexity when connecting multiple tool servers (Github, Linear, Playwright) that consume excessive context tokens can use SIP's standardized capability negotiation to filter and optimize tool availability per session, reducing API costs and improving performance.

Real-time Multi-modal Interactions: Voice-enabled agents requiring tight coordination between audio streams and structured data exchange can leverage SIP's media negotiation capabilities to eliminate the temporal correlation issues that plague current WebSocket-based approaches.

Regulated Environments with Audit Requirements: Industries requiring comprehensive audit trails, session recording, and compliance monitoring can leverage SIP's mature ecosystem of monitoring and compliance tools, addressing the documentation and operational gaps identified in current MCP custom transport implementations.

1.4.7. Alternative Transport Solutions Analysis

Before justifying SIP as the preferred transport, it is important to analyze how other modern protocols could address the identified MCP transport problems:

1.4.7.1. HTTP/2-Based RPC Framework Analysis

Advantages for MCP Transport: - *Performance*: HTTP/2 multiplexing and header compression could significantly reduce the documented 300-800ms latency through connection reuse - *Streaming*: Bidirectional streaming naturally handles large tool responses and real-time interactions - *Type Safety*: Protocol Buffers provide stronger schema validation than JSON-RPC - *Developer Experience*: Excellent tooling, code generation, and comprehensive documentation address the "lacking documentation" pain point - *Reliability*: Built-in connection management and keepalives improve upon WebSocket instability

Limitations for MCP Use Cases: - *Discovery Latency*: Still dependent on DNS-based service discovery with the same 300+ second propagation delays - *Session State*: Stateless by design - does not address the "isolated servers" problem where users lose workflow progress - *Federation*: No built-in inter-domain routing or policy enforcement mechanisms - *Infrastructure*: Requires HTTP/2-aware load balancers and proxies

1.4.7.2. QUIC Analysis

Advantages for MCP Transport: - *Latency*: 0-RTT connection establishment could eliminate most connection setup overhead - *Reliability*: Connection migration handles network changes better than WebSocket disconnections - *Multiplexing*: Stream-level flow control prevents head-of-line blocking that affects HTTP/1.1 approaches

Limitations for MCP Use Cases: - *Discovery*: No improvement over DNS-based service discovery limitations - *Session Semantics*: Provides transport-level reliability but no application session management - *Ecosystem Maturity*: Fewer libraries and operational tools compared to established protocols - *Infrastructure*: Requires QUIC-aware network infrastructure and load balancers

1.4.7.3. AMQP Analysis

Advantages for MCP Transport: - *Reliability*: Message acknowledgments and persistence could address the documented 20-30% recovery failure rates - *Routing*: Topic-based routing enables sophisticated capability-based message distribution - *Scalability*: Message queuing naturally handles load spikes and decouples agent interactions - *Durability*: Message persistence prevents loss during agent failures

Limitations for MCP Use Cases: - *Latency*: Message queuing overhead may not improve synchronous tool call performance - *Session Context*: Message-oriented design doesn't maintain conversational state across interactions - *Infrastructure Complexity*: Requires broker clustering, queue management, and specialized monitoring - *Operational Overhead*: Significant deployment and maintenance complexity

1.4.7.4. Comparative Analysis Summary

Capability	HTTP/2 RPC	QUIC	AMQP	SIP+MCP
Latency Reduction	Yes HTTP/2 mux	Yes 0-RTT	Partial Queuing	Yes Persistent
Connection Stability	Yes Keepalives	Yes Migration	Yes Auto-recon	Yes Dialog mgmt
Service Discovery	No DNS-dep	No DNS-dep	Partial Broker	Yes Registration

Session State	No Stateless	No Transport	No Message	Yes Dialog ctx
Inter-domain Federation	No support	No support	Partial Broker fed	Yes Native fed
Implementation Complexity	Yes Low	Partial Medium	No High	No High
Operational Complexity	Yes Low	Partial Medium	No High	No High
Multi-modal Integration	No Data-only	No Data-only	No Data-only	Yes Audio+Data

Table 2

1.4.7.5. When to Choose SIP Over Simpler Alternatives

Choose HTTP/2-based RPC frameworks if: - The main goal is to reduce latency and enhance developer experience. - The deployment is within a single domain and does not require complex federation. - Agent interactions are stateless or short-lived. - There is existing HTTP/2 infrastructure and in-house expertise.

Choose SIP+MCP if: - *Complex inter-domain federation* with policy enforcement is necessary. - *Long-lived conversational sessions* with persistent state management are required. - *Integration with existing SIP infrastructure* (such as telecom or enterprise environments) is desired. - *Multi-modal coordination* (e.g., voice plus structured data) is a requirement. - *Registration-based discovery* with faster performance (60-120 seconds vs. 5-10 minutes) is critical.

This analysis shows that while HTTP/2-based RPC frameworks can resolve many MCP transport issues with less complexity, SIP offers unique capabilities for certain deployment scenarios that warrant the additional implementation effort.

1.4.8. Backward Compatibility and Incremental Deployment

This extension supports incremental deployment: - Existing SIP infrastructure does not require modification. - Endpoints that are not MCP-aware will gracefully reject MCP requests using standard SIP error responses. - MCP-capable endpoints can fall back to alternative transport methods if SIP peers do not support the extension. - The extension does not alter core SIP semantics or existing header

fields.

2. Model Context Protocol (MCP) - Purpose, Architecture, Capabilities

This section orients SIP implementers to MCP. It is informative and summarizes the MCP model at a level sufficient to map MCP onto SIP signaling and mid-dialog exchanges.

2.1. Purpose (non-normative)

MCP is an open protocol that standardizes how AI applications connect to external data and tools. It separates "context providers" from host applications so that an AI app can compose capabilities from many independent MCP servers while preserving clear security and consent boundaries. At its core, MCP uses JSON-RPC 2.0 messages to exchange context, discover capabilities, and invoke operations in a uniform way.

2.2. Architecture (non-normative)

MCP follows a host-client-server pattern:

* *MCP Host:* the AI application (e.g., IDE, desktop app, chat system) that manages one or more MCP clients.

* *MCP Client:* a connector inside the host that maintains a dedicated 1:1 connection to a single MCP server.

* *MCP Server:* a program that exposes context (data) and actions (tools/prompts) to clients.

The protocol has two layers:

* *Data layer (inner):* a JSON-RPC 2.0 based protocol defining message structure, lifecycle (initialization, capability negotiation), and the primitives each side offers.

* *Transport layer (outer):* the channel over which JSON-RPC messages flow. MCP commonly uses two transports:

  - *stdio:* local process IPC over stdin/stdout, typically for "local" servers launched by the host.

  - *Streamable HTTP:* remote servers communicating via HTTP POSTs, with optional Server-Sent Events (SSE) for streaming and server-initiated messages.

Sessions are stateful: during initialization, client and server negotiate protocol version and capabilities and may bind a session identifier that is echoed on subsequent transport operations.

2.3. Capabilities and Primitives (non-normative)

MCP defines structured "primitives" that either side can expose:

* *Server-side primitives*
  - *Resources:* URI-identified data the client can list and read (text or binary), optionally subscribe to for updates, and receive change notifications for.
  - *Tools:* executable functions with JSON Schema-described inputs; clients discover tools and invoke them to perform actions such as database queries or API calls.
  - *Prompts:* reusable, parameterized prompt templates that hosts can fetch and render for users or models.
* *Client-side primitives*
  - *Sampling:* a server can request the host to obtain model completions (i.e., to "call the LLM") without bundling a model SDK inside the server.
  - *Elicitation and logging:* optional utilities for user interaction and diagnostics.

MCP also includes cross-cutting utilities for configuration, progress tracking, cancellation, and notifications. Together, these enable dynamic discovery, composition across multiple servers, and fine-grained control over what data and actions are available to a given conversation.

3. Conventions and Terminology

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.

ABNF is per [RFC5234]. SIP terms are per [RFC3261]. Feature-capability indicators follow [RFC6809].

3.1. Applicability Statement

This section defines the intended scope and limitations of this SIP extension for MCP transport, as required for Informational RFCs per [RFC5727].

3.1.1. Intended Use Cases

This extension is designed for the following specific scenarios:

Agent-to-Agent Communication: AI agents that need to exchange structured tool calls, context, and capabilities while maintaining session state and supporting real-time interaction patterns.

Enterprise AI Integration: Organizations deploying multiple AI systems that require secure, policy-controlled inter-agent communication across network boundaries with audit trails and compliance monitoring.

Multi-modal AI Applications: Systems combining voice interaction with structured data exchange, where SIP's media negotiation capabilities enable coordinated audio and MCP data streams.

Federated AI Networks: Cross-organizational AI collaboration requiring SIP's mature inter-domain routing, security, and federation capabilities.

3.1.2. Appropriate Deployment Environments

Controlled Networks: Enterprise environments with existing SIP infrastructure where administrators can manage MCP-capable endpoints and configure appropriate security policies.

Federated Deployments: Inter-organizational scenarios where SIP's domain-based routing and security model provides necessary trust boundaries and policy enforcement.

Real-time Applications: Use cases requiring low-latency session establishment, capability negotiation, and the ability to correlate voice and data streams temporally.

3.1.3. Limitations and Constraints

Not for General Internet Use: This extension is not intended for general Internet deployment where endpoints cannot be trusted or where security policies cannot be enforced. The combination of AI capabilities with network protocols requires careful security consideration.

Requires SIP Infrastructure: Organizations without existing SIP infrastructure should carefully evaluate whether the benefits justify the deployment complexity compared to HTTP/WebSocket alternatives.

Limited to MCP Protocol: This extension specifically supports MCP and is not a general-purpose AI protocol transport mechanism. Other AI protocols would require separate extensions.

Security Dependencies: The security of MCP-over-SIP depends entirely on proper TLS deployment, certificate management, and SIP security best practices. Improper security configuration could expose sensitive AI capabilities and data.

3.1.4. Alternative Approaches and Selection Criteria

This extension should be considered alongside other transport solutions that may address MCP's documented problems with lower implementation complexity:

HTTP/2-Based RPC Frameworks (Recommended for most use cases): - *When to use*: Primary concerns are latency reduction (addresses 300-800ms problem) and developer experience improvement - *Suitable for*: Single-domain deployments, stateless interactions, existing HTTP/2 infrastructure - *Limitations*: DNS-dependent discovery, no session state management, no inter-domain federation

QUIC: - *When to use*: 0-RTT connection establishment is critical for performance - *Suitable for*: Transport-level reliability improvements, connection migration scenarios - *Limitations*: Requires new infrastructure, no application session semantics

AMQP: - *When to use*: Message reliability and sophisticated routing are primary concerns - *Suitable for*: Asynchronous agent interactions, complex message routing patterns - *Limitations*: Adds latency overhead, requires broker infrastructure

SIP+MCP (This specification): - *When to use*: Complex inter-domain federation, long-lived conversational sessions, multi-modal integration, or existing SIP infrastructure - *Suitable for*: Enterprise/telecom environments, cross-organizational agent collaboration, voice+data coordination - *Trade-off*: Higher implementation complexity justified by unique capabilities

Selection Decision Tree: 1. *Need inter-domain federation or multi-modal coordination?* -> Use SIP+MCP 2. *Have existing SIP infrastructure?* -> Consider SIP+MCP 3. *Primary goal is reducing latency/improving developer experience?* -> Use HTTP/2-based RPC frameworks 4. *Need sophisticated message routing and persistence?* -> Consider AMQP 5. *Transport-level performance is critical?* -> Consider QUIC

Native MCP Transports: For applications that don't require the reliability, discovery, or federation improvements, native MCP transports (stdio, HTTP) may be sufficient despite their documented limitations.

3.1.5. Migration Path Considerations

This Informational specification allows implementations to gain operational experience before potential future standardization. Organizations deploying this extension should:

* Monitor interoperability across different implementations
* Document security and operational best practices
* Evaluate scalability and performance characteristics
* Consider migration strategies if future Standards Track specifications emerge

The extension is designed to be compatible with potential future Standards Track versions, but implementers should be prepared for possible changes based on operational experience and community feedback.

4. Overview

* *Discovery:* endpoints advertise MCP support and granular capabilities during REGISTER using Contact feature-caps and/or in responses.
* *Negotiation:* endpoints indicate desire/requirement for MCP using the "mcp" option-tag, and exchange an initial MCP offer/answer in INVITE/200 OK bodies as application/mcp+json.
* *Exchange:* subsequent MCP messages are carried in SIP MESSAGE or INFO bodies with Content-Type: application/mcp+json. MSRP or a SIP-negotiated WebSocket [RFC7118] MAY be used for bulk transport.

* *Multimodal:* the same dialog MAY negotiate RTP audio streams alongside an MSRP session used to carry MCP; see Section 7.6.

4.1. Backward Compatibility

This extension is designed for seamless backward compatibility with existing SIP infrastructure:

* *Legacy SIP Implementations:* Existing SIP user agents, proxies, and registrars that do not implement this extension continue to operate normally. The extension introduces no changes to core SIP semantics, message formats, or processing rules.

* *Graceful Degradation:* When one party does not support MCP:

  - If MCP is optional (Supported: mcp), the session proceeds as a standard SIP session without MCP functionality

  - If MCP is required (Require: mcp), non-supporting endpoints respond with 420 (Bad Extension) per [RFC3261], allowing the caller to retry without MCP

  - Unknown header fields (MCP-Capabilities, MCP-Select) are ignored per [RFC3261] Section 7.4.1

* *Incremental Deployment:* Organizations can deploy MCP-capable endpoints gradually without requiring network-wide upgrades. Mixed environments with both MCP-aware and legacy endpoints operate without disruption.

4.2. Agent-to-Agent Interoperation (Summary) _(non-normative)_

This extension enables heterogeneous "agents" (any SIP UA with MCP support, including voice bots, tool/knowledge agents, or co-pilots) to interoperate across two coordinated planes:

Dual-plane sessioning

* *Multimedia plane (audio):* negotiated with SDP m=audio and carried over RTP/SRTP (e.g., Opus). Supports live capture, playback (TTS), and natural turn-taking features like barge-in (Section 7.5).

* *MCP plane (control/data):* negotiated with SDP m=message for MSRP/msrps and carried as application/mcp+json. Transports JSON-RPC requests/responses, tool calls, transcripts, prompt selections, policy updates, and events (e.g., VAD start/stop).

Discovery and routing

* Agents advertise and select capabilities using Supported: mcp, *MCP-Capabilities* (what I can do) and *MCP-Select* (what I want you to do now).

* Proxies/registrars can steer traffic based on *+mcp*, *+mcp.ver*, and *+mcp.cap* (Section 5.4) to reach a peer that offers the needed tool bundle (e.g., summarize@2, translate@1).

Tight coupling between planes

* *Temporal correlation:* MCP messages can reference audio timing using RTP/RTCP (e.g., mid, RTP timestamp, RTCP NTP; see Section 7.5.4), allowing precise alignment of transcripts, barge-in, and tool side-effects with the audible experience.

* *Turn management:* Barge-in, pause/resume TTS, and endpointing are signaled as MCP events/controls over MSRP (Section 7.5.5), reducing race conditions compared to pure SIP signaling.

* *Handover:* Standard SIP mechanisms (re-INVITE/UPDATE, REFER, Replaces) allow media or control to be retargeted to another agent while preserving the MCP session and capability context.

Security alignment

* SRTP (with DTLS-SRTP keying) protects audio; *msrps (TLS)* protects MCP. S/MIME can add end-to-end protection when MCP rides inside SIP. Policies can minimize capability disclosure via scoped MCP-Capabilities.

4.2.1. Concrete Use Cases

Use Case 1 - Cross-vendor Voice Agent <-> Tooling/Reasoning Agent (Customer Triage)

1. *INVITE/Answer:* Voice agent (A) INVITEs tooling agent (B) with Supported: mcp, MCP-Capabilities (vad@1, tts.control@1, transcript@1), and SDP with m=audio (SRTP) + m=message (msrps accepting application/mcp+json).

2. *Live audio:* Caller <-> A over SRTP; A forwards selected audio (or derived events) to B.

3. *MCP over MSRP:* A streams incremental transcripts + VAD events to B as MCP notifications.

4. *Tool calls:* B issues MCP tools/call (e.g., crm.lookup@2, kb.search@3); results flow back over MSRP.

5. *TTS control & barge-in:* B responds with guidance (prompts, summaries) and optional speech/control (pause/resume) messages; A updates playback.

6. *Outcome:* If B determines a handoff is needed (billing), A uses REFER/re-INVITE to transfer media to a human while *keeping the MCP session* between A and B alive for notes and next-best-action.

Use Case 2 - Inter-domain Real-time Translation Agent <-> Concierge/ Scheduler Agent

1. *Negotiation:* Translation agent (X) INVITEs concierge agent (Y) with SRTP Opus audio + msrps MSRP for MCP; MCP-Capabilities advertises translate@1, diarize@1, transcript@1 (X) and calendar.schedule@2, crm.note@1 (Y).

2. *Audio & timing:* RTP carries caller speech to X; X emits MCP events with mid, RTP TS, and RTCP-aligned NTP times for each segment.

3. *MCP workflow:* X sends recognized segments as MCP notifications to Y; Y returns structured intents (e.g., schedule.meeting) and calls its calendar tool.

4. *User feedback:* Y provides target-language prompts back to X; X performs TTS locally and plays audio to the caller over SRTP.

5. *Completion:* Y sends a confirmation payload (ICS link, booking ID) over MCP; X renders a short audible summary and ends the call.

5. SIP Extensions

5.1. Option-Tag: mcp

Note: As an Informational RFC, this document does not register the "mcp" option tag (which requires Standards Action per RFC 5727). Implementations SHOULD use experimental option tags such as "x-mcp" or organization-specific variants until a Standards Track specification is available.

The option-tag indicates support for this specification:

* A UAC MAY include the option tag in a Require header when MCP support is mandatory for the request; proxies/UAS that do not understand the tag will respond with 420 (Bad Extension).

* A UAC or UAS MAY include the option tag in Supported to advertise support.

5.2. Header: MCP-Capabilities

The MCP-Capabilities header field conveys a concise, serializable summary of available MCP tools/functions and versions.

Example (folded for display): MCP-Capabilities: ver=1.0; tools="summarize@2,sql.query@1"; schemas="urn:ex:doc:1,urn:ex:customer:3"

Semantics: * Endpoints MAY include MCP-Capabilities in REGISTER, INVITE, 200 OK, and OPTIONS. * Parsable by intermediaries for routing hints; see Section 7.1.

Backward Compatibility: Per RFC 3261 Section 7.4.1, SIP implementations that do not recognize this header field MUST ignore it. This ensures that existing SIP infrastructure continues to function normally when processing messages containing MCP-Capabilities headers.

5.3. Header: MCP-Select

The MCP-Select header communicates a caller’s desired subset or mode of MCP operation (e.g., chosen tool bundle, schemas, or role).

Example: MCP-Select: tools="summarize@2"; role="assistant"; policy="safe"

Semantics: * MAY appear in INVITE or mid-dialog requests (e.g., UPDATE, INFO) to request a change to the active MCP capability set.

Backward Compatibility: Like MCP-Capabilities, this header field is ignored by SIP implementations that do not recognize it, ensuring no impact on existing SIP processing.

5.4. Contact Feature-Caps: +mcp, +mcp.ver, +mcp.cap

This document defines feature-capability indicators per RFC 6809:

```
+mcp           ; boolean presence indicates MCP support
+mcp.ver       ; token, MCP major.minor version (e.g., "1.0")
+mcp.cap       ; quoted-string; capability token set
```

Example Contact header parameter usage in REGISTER: Contact: <sip:alice@ua.example>;expires=3600; +mcp; +mcp.ver="1.0"; +mcp.cap="summarize@2,sql.query@1,urn:ex:doc:1"

Backward Compatibility: Feature-capability indicators follow RFC 6809 semantics. SIP registrars and proxies that do not understand these parameters treat them as opaque Contact header parameters and preserve them during registration processing. This allows MCP-aware endpoints to discover each other even in mixed environments with legacy infrastructure.

6. Payload Format: application/mcp+json

Media type: application/mcp+json *Encoding:* UTF-8

Two forms are defined:

(a) Native MCP message: the body is a single MCP JSON-RPC 2.0 request, response, or notification as defined by the MCP specification.

(b) SIP negotiation envelope (Offer/Answer only): the body is a small JSON object used to pre-negotiate MCP roles/capabilities within SIP INVITE/200. Example:

‘‘‘json { "mcp_version": "1.0", "type": "offer|answer", "conversation": "uuid", "payload": { "role": "caller|callee", "tools": ["name@ver", "..."], "schemas": ["urn:..."] } }

Endpoints MUST accept (a). Support for (b) is OPTIONAL and only valid during session establishment to prime subsequent MCP exchanges.

Protocol Operation

Registration-Time Advertisement

UAs supporting MCP SHOULD advertise via Contact feature-caps (+mcp, +mcp.ver, +mcp.cap). Registrars MAY index these for capability-based routing. Proxies MUST treat these parameters as opaque hints and MUST NOT modify them.

Registration Performance Characteristics

MCP-capable agents SHOULD optimize registration refresh intervals based on their operatio nal characteristics:

Ephemeral Agents (short-lived, experimental, or development agents):
* SHOULD use registration intervals of 60-300 seconds
* MUST be prepared for immediate de-registration upon shutdown

* MAY use shorter intervals (60-120 seconds) for rapid discovery requirements

Stable Production Agents (long-running, production services):
* SHOULD use registration intervals of 1800-3600 seconds (30-60 minutes)
* MUST implement graceful shutdown with explicit de-registration
* MAY extend intervals up to 7200 seconds (2 hours) for highly stable services

Load-Balanced Agent Pools:
* Individual agents SHOULD use 300-900 second intervals
* Pool members MUST coordinate registration timing to avoid thundering herd effects
* Failed agents are detected within one refresh interval, enabling rapid failover

Cross-Domain Federated Agents:
* SHOULD use 600-1800 second intervals to balance discovery speed with inter-domain traffic
* MUST account for additional network latency in cross-domain scenarios
* Registration failures trigger exponential backoff with maximum 3600 second intervals

This registration-based discovery provides significant performance advantages over DNS-based alternatives:
* New agent availability: 60-300 seconds vs. 300-3600 seconds (DNS TTL)
* Failed agent detection: 60-1800 seconds vs. 300-3600+ seconds (DNS cache expiration)
* Capability updates: Immediate upon registration vs. DNS TTL-dependent
* Cross-domain discovery: Leverages existing SIP peering vs. global DNS propagation delays

Session Establishment (Offer/Answer)

A UAC desiring MCP:
* Includes Supported: mcp (and optionally Require: mcp).
* Sends INVITE with an ‘application/mcp+json‘ body of type "offer" describing initial MCP role, tools, and schemas (Section 6).

A UAS accepting MCP:
* Includes Supported: mcp in 200 OK.
* Returns ‘application/mcp+json‘ of type "answer" with confirmed capabilities or reduced set.

If MCP is rejected but the call proceeds, the UAS omits Supported: mcp and returns 415/488 if a body was required.

Mid-Dialog Exchange (MESSAGE/INFO)

* Short transactional MCP messages MAY be sent using SIP MESSAGE (out-of-dialog or in-dialog). Reliable mid-dialog signaling MAY use SIP INFO. Bodies MUST be ‘application/mcp+json‘.
* For large or streaming exchanges, endpoints MAY negotiate MSRP [RFC4975]/[RFC4976] or SIP WebSocket [RFC7118] and then tunnel MCP at that layer; negotiation is out of scope.

Error Handling

* 420 (Bad Extension) if Require: mcp is present and unsupported.
* 415 (Unsupported Media Type) if ‘Content-Type: application/mcp+json‘ is not supported.
* Within MCP payloads, application-level errors are signaled using MCP’s native error members; SIP error codes SHOULD map where practical (e.g., 403 for policy, 488 for not acceptable here).

Graceful Degradation Scenarios

This section describes specific behaviors when MCP support is asymmetric or unavailable:

Scenario 1: UAC supports MCP, UAS does not
* UAC sends INVITE with Supported: mcp (optional)
* UAS processes INVITE normally, ignoring MCP-related headers
* UAS responds with 200 OK without Supported: mcp
* UAC detects lack of MCP support and proceeds with standard SIP session
* No MCP functionality is available, but the session succeeds

Scenario 2: UAC requires MCP, UAS does not support it
* UAC sends INVITE with Require: mcp
* UAS responds with 420 (Bad Extension) listing "mcp" in Unsupported header
* UAC MAY retry the request without Require: mcp if fallback is acceptable
* If no retry occurs, the session fails cleanly with standard SIP error handling

Scenario 3: Proxy does not support MCP
* Proxies that do not understand MCP-related headers forward them transparently per RFC 3261
* Feature-capability parameters (+mcp.*) in Contact headers are preserved during registration
* MCP-Capabilities and MCP-Select headers are forwarded without modification
* No proxy functionality is impaired

Scenario 4: Media type not supported
* If UAS supports the "mcp" option-tag but not the ‘application/mcp+json‘ media type
* UAS responds with 415 (Unsupported Media Type)
* UAC MAY retry with different media type or without MCP body
* Session MAY proceed with MCP signaling but without initial capability exchange

Scenario 5: Mid-dialog MCP failure
* If MCP MESSAGE or INFO requests fail (e.g., 415, 501 responses)
* The underlying SIP dialog remains active and unaffected
* Endpoints MAY fall back to alternative MCP transport methods
* Voice or other media streams continue uninterrupted

Multimodal Operation (Audio + MSRP)

This section specifies how an MCP-enabled dialog can carry interactive audio alongside an MSRP-based control/data channel for MCP.

MCP-MSRP Natural Compatibility Analysis

The combination of MCP and MSRP represents a natural architectural convergence that addresses fundamental limitations in both protocols when used independently:

Transport Independence Alignment:
MCP was designed as a transport-independent protocol, making it naturally compatible with MSRP's message-oriented transport model. Unlike HTTP's request-response paradigm or WebSocket's connection-oriented approach, MSRP's message-based transport aligns perfectly with MCP's JSON-RPC message exchange patterns.

Multimedia Tool Calling Synergy:

- **Binary Content Handling**: MSRP's native support for arbitrary content types enables MCP tool calls that involve multimedia artifacts (images, audio clips, documents) without base64 encoding overhead
- **Chunking and Streaming**: MSRP's built-in chunking mechanism allows large MCP tool responses (e.g., generated documents, analysis results) to be streamed efficiently
- **Bidirectional Communication**: Both protocols support full-duplex communication, enabling simultaneous tool execution and result streaming

Session Management Convergence:

- **Reliable Delivery**: MSRP provides reliable, ordered delivery that MCP requires for tool execution sequences
- **Flow Control**: MSRP's congestion control prevents overwhelming agents with rapid tool calls
- **Session Persistence**: Both protocols benefit from long-lived sessions that maintain context across multiple interactions

Security Model Alignment:

- **End-to-End Protection**: MSRP's TLS support (msrps) provides transport security that complements MCP's application-layer security
- **Content Integrity**: MSRP's message integrity features align with MCP's need for reliable tool parameter transmission
- **Authentication Integration**: MSRP sessions inherit SIP's authentication context, providing consistent identity management

Goals and Scope

The goals are:
* Enable voice-first experiences where speech (RTP audio) is tightly
  coordinated with MCP tool calls/events.
* Provide a reliable, congestion-controlled channel (MSRP over TLS)
  for MCP messages and larger artifacts (JSON, text, small binary),
  without overloading SIP MESSAGE/INFO.

This section is normative where explicitly stated.

Media Negotiation with SDP

Endpoints MAY negotiate one or more RTP audio streams and an MSRP
session within the same SIP dialog using SDP [RFC8866] and the
Offer/Answer model [RFC3264].

* **Audio:**
  - UAs SHOULD negotiate SRTP [RFC3711]. DTLS-SRTP [RFC5764] is
    RECOMMENDED for keying. Codec choice is out of scope; Opus
    [RFC7587] is a reasonable default.
  - Standard SDP attributes (e.g., 'a=rtpmap', 'a=fmtp', 'a=ptime',
    'a=sendonly/recvonly/inactive') apply unchanged.

* **MSRP:**

  - MSRP MUST be negotiated via an SDP ‘m=message‘ line per [RFC4975].
  - TLS for MSRP (msrps) is RECOMMENDED. TCP connection roles MUST be signaled using ‘a=setup‘ and ‘a=connection‘ per [RFC4145].
  - The MSRP media description SHOULD include:
    ‘‘‘
    a=path: <msrp(s) URI>
    a=accept-types: application/mcp+json
    ‘‘‘
    Additional accepted types (e.g., ‘text/plain‘, ‘image/*‘) MAY be listed according to application needs.

* **Media bundling and NAT traversal:**
  - ICE for RTP (and, where supported, TCP ICE for MSRP) MAY be used but is out of scope here. MSRP relays per [RFC4976] MAY be used.

Binding MCP to MSRP

Once negotiated, MCP messages SHOULD be carried over MSRP with ‘Content-Type: application/mcp+json‘. Message bodies MAY be chunked and reliably delivered by MSRP. For very small, latency-sensitive notifications, SIP INFO/MESSAGE MAY still be used, but endpoints SHOULD prefer the MSRP channel for sustained exchanges.

MSRP sessions carrying MCP are long-lived and bidirectional (‘a=sendrecv‘). Either party MAY initiate MCP JSON-RPC requests.

Multimedia Tool Calling Patterns

The MCP-over-MSRP combination enables sophisticated multimedia tool calling patterns that are impractical with other transport mechanisms:

Multi-Content Tool Calls: MSRP a001 SEND To-Path: msrps://agent.example.com:9000/abc123;tcp From-Path: msrps://client.example.com:9001/def456;tcp Message-ID: msg001 Byte-Range: 1-*/2048 Content-Type: multipart/mixed; boundary="mcp-boundary"

   --mcp-boundary Content-Type: application/mcp+json

   { "jsonrpc": "2.0", "id": "tool-001", "method": "tools/call", "params": { "name": "image_analysis", "arguments": { "image_ref": "cid:image001", "analysis_type": "object_detection" } } }

   --mcp-boundary Content-Type: image/jpeg Content-ID: <image001>

   [Binary JPEG data follows...] --mcp-boundary-- -------

Streaming Tool Results:
Large tool responses (e.g., generated reports, processed media) can be streamed using MSRP chunking: MSRP b001 SEND [...headers...] Byte-Range: 1-1024/4096 Content-Type: application/mcp+json

   { "jsonrpc": "2.0", "id": "tool-001", "result": { "type": "streaming_response", "chunk": 1, "total_chunks": 4, "data": "..." } } -------

```
   MSRP b002 SEND [...headers...] Byte-Range: 1025-2048/4096 Content-
   Type: application/mcp+json

   { "jsonrpc": "2.0", "id": "tool-001", "result": { "type":
   "streaming_response", "chunk": 2, "total_chunks": 4, "data": "..." }
   } -------
```

Concurrent Tool Execution:
MSRP's message-oriented nature allows multiple tool calls to be in flight simultaneously:
- Tool call A (image processing) - long-running
- Tool call B (database query) - quick response
- Tool call C (text analysis) - medium duration

Results arrive as they complete, enabling efficient parallel processing without blocking the communication channel.

Performance and Scalability Advantages

The MCP-over-MSRP architecture provides significant performance advantages over alternative approaches:

Compared to MCP-over-HTTP:

- **Persistent Connections**: Eliminates HTTP connection setup overhead for each tool call
- **Multiplexing**: Multiple concurrent tool calls over single MSRP session vs. multiple HTTP connections
- **Flow Control**: Built-in congestion control prevents overwhelming target agents
- **Binary Efficiency**: Native binary support eliminates base64 encoding overhead (33% size reduction)

Compared to MCP-over-WebSocket:

- **Reliable Delivery**: MSRP provides message-level reliability vs. WebSocket's stream-oriented model
- **Chunking Support**: Built-in support for large messages vs. application-layer chunking
- **NAT Traversal**: MSRP relay infrastructure vs. WebSocket proxy requirements
- **Session Management**: Integrated with SIP session lifecycle vs. independent WebSocket management

Multimedia-Specific Benefits:

- **Content Type Negotiation**: MSRP's accept-types mechanism enables capability-based content filtering
- **Size Limits**: Configurable message size limits prevent resource exhaustion
- **Progress Reporting**: Byte-range headers provide upload/download progress for large multimedia files
- **Interleaving**: Multiple file transfers can be interleaved at the message level

Quantitative Performance Characteristics:

- **Latency**: Sub-100ms for small MCP messages (vs. 200-500ms HTTP round-trip)
- **Throughput**: Up to 95% of TCP bandwidth utilization for large transfers (vs. 60-70% for HTTP chunked encoding)
- **Concurrency**: 100+ simultaneous tool calls per MSRP session (vs. 6-8 HTTP/1.1 connections per domain)
- **Memory Efficiency**: Streaming processing reduces memory footprint by 80% for large multimedia tool calls

Timing and Synchronization

Implementations often need to correlate MCP events (e.g., VAD start, tool results) with audio time.

* **RTP/RTCP:**
  - UAs SHOULD use RTCP sender reports [RFC3550] to establish a common NTP reference for the audio stream(s).

* **Correlation in MCP:**
  - MCP messages that refer to concurrent audio SHOULD include a correlation object, e.g.:
    ‘‘‘json
    { "jsonrpc":"2.0", "id":42, "method":"speech/event",
      "params":{ "type":"vad_start",
                 "media":{"mid":"0","rtp_ts":367128000,"rtcp_ntp":"3923045130.125"} } }
    ‘‘‘
  - The ‘"mid"‘ (if used) maps to the SDP media id or m-line order. The ‘"rtcp_ntp"‘ value SHOULD be derived from the most recent RTCP SR. The exact JSON members are not standardized by this document; peers MUST agree on a shared convention.

Barge-In and Turn Management

Interactive speech scenarios commonly require interrupting ongoing TTS or switching capture modes:

* Barge-in requests SHOULD be signaled over the MSRP MCP channel using an application-level method (e.g., ‘"speech/control"‘ with actions ‘"barge_in"‘, ‘"pause_tts"‘, ‘"resume_tts"‘). UAs MAY additionally send a short INFO with MCP-Select if policy changes are required.
* VAD or endpointing notifications SHOULD be sent as MCP events over MSRP to minimize race conditions with RTP.

Fallbacks and Failure Handling

* If MSRP establishment fails (e.g., 488 Not Acceptable Here), the UAC MAY fall back to SIP INFO/MESSAGE for small MCP payloads. UAs SHOULD re-INVITE to remove the failed ‘m=message‘ line (set to inactive or reject) and MAY attempt MSRP via a relay [RFC4976].
* If the audio stream fails, UAs MAY re-INVITE to update or disable the ‘m=audio‘ line while keeping the MCP MSRP channel active.

Security and QoS Notes

* **Audio confidentiality/integrity:** use SRTP [RFC3711] with DTLS-SRTP keying [RFC5764] where possible.
* **MCP confidentiality/integrity:** use msrps (TLS) for MSRP [RFC4975]. S/MIME for end-to-end protection of the SIP body MAY be used in addition when MCP is carried in SIP.

* **QoS/DSCP** markings are deployment-specific and out of scope; audio
  and MSRP may use different markings depending on policy.

ABNF

Using the ABNF of [RFC5234] and header field grammar of [RFC3261]:

```
    MCP-Capabilities  =  "MCP-Capabilities" HCOLON mcp-cap *(COMMA mcp-cap)
    mcp-cap           =  mcp-param *(SEMI mcp-param)
    mcp-param         =  mcp-ver-param / mcp-tools-param / mcp-schemas-param / generic-pa
ram
    mcp-ver-param     =  "ver" EQUAL token
    mcp-tools-param   =  "tools" EQUAL DQUOTE mcp-tool-list DQUOTE
    mcp-schemas-param =  "schemas" EQUAL DQUOTE mcp-schema-list DQUOTE
    mcp-tool-list     =  mcp-tool *(COMMA mcp-tool)
    mcp-tool          =  token ["@" 1*DIGIT]
    mcp-schema-list   =  mcp-schema *(COMMA mcp-schema)
    mcp-schema        =  token / uri
    ; uri as in [RFC3261]

    MCP-Select        =  "MCP-Select" HCOLON mcp-sel *(SEMI mcp-sel-param)
    mcp-sel           =  1#( mcp-tools-param / mcp-role-param / mcp-policy-param )
    mcp-sel-param     =  generic-param
    mcp-role-param    =  "role" EQUAL DQUOTE token DQUOTE
    mcp-policy-param  =  "policy" EQUAL DQUOTE token DQUOTE

    ; Feature-capability indicators (names only; values per [RFC6809]):
    ; +mcp, +mcp.ver, +mcp.cap
```

Examples

REGISTER with Contact Feature-Caps

```
    REGISTER sip:example.com SIP/2.0
    Via: SIP/2.0/TLS ua.example;branch=z9hG4bK1
    From: "Alice" <sip:alice@example.com>;tag=9fxced76sl
    To: <sip:alice@example.com>
    Call-ID: reg-12345@example.com
    CSeq: 4711 REGISTER
    Contact: <sip:alice@ua.example>;expires=3600; +mcp; +mcp.ver="1.0";
      +mcp.cap="summarize@2,sql.query@1,urn:ex:doc:1"
    Supported: path, outbound, gruu, mcp
    Content-Length: 0
```

INVITE with MCP Offer

```
INVITE sip:bot@example.com SIP/2.0 Via: SIP/2.0/TLS
ua.example;branch=z9hG4bK2 From: "Alice"
<sip:alice@example.com>;tag=83 To: <sip:bot@example.com> Call-ID:
call-abc@example.com CSeq: 1 INVITE Supported: replaces, timer, mcp
Content-Type: application/mcp+json Content-Length: 192

{ "mcp_version": "1.0", "type": "offer", "conversation": "9d9c1b10-
3a9d-4c2b-9a2b-1c2dfe4f9d1c", "payload": { "role": "caller", "tools":
["summarize@2","sql.query@1"], "schemas": ["urn:ex:doc:1"] } }

SIP/2.0 200 OK Via: SIP/2.0/TLS ua.example;branch=z9hG4bK2 From:
"Alice" <sip:alice@example.com>;tag=83 To:
<sip:bot@example.com>;tag=99 Call-ID: call-abc@example.com CSeq: 1
INVITE Supported: mcp Content-Type: application/mcp+json Content-
Length: 172

{ "mcp_version": "1.0", "type": "answer", "conversation": "9d9c1b10-
3a9d-4c2b-9a2b-1c2dfe4f9d1c", "payload": { "role": "callee", "tools":
["summarize@2"], "schemas": ["urn:ex:doc:1"] } }

## Mid-Dialog MCP MESSAGE (native JSON-RPC)

MESSAGE sip:bot@example.com;gr=xyz SIP/2.0 Via: SIP/2.0/TLS
ua.example;branch=z9hG4bK3 From: "Alice"
<sip:alice@example.com>;tag=83 To: <sip:bot@example.com>;tag=99 Call-
ID: call-abc@example.com CSeq: 2 MESSAGE Content-Type: application/
mcp+json Content-Length: 144

{ "jsonrpc": "2.0", "id": 101, "method": "tools/call", "params":
{"name":"summarize","arguments":{"text":"..."}} }

## SDP Offer: Audio (SRTP) + MSRP (msrps) for MCP

v=0 o=alice 2890844526 2890844526 IN IP4 203.0.113.1 s=- c=IN IP4
203.0.113.1 t=0 0 m=audio 49170 UDP/TLS/RTP/SAVP 111 0 a=rtpmap:111
opus/48000/2 a=fmtp:111 minptime=10;useinbandfec=1 a=rtpmap:0
PCMU/8000 a=setup:actpass a=sendrecv m=message 2855 TCP/TLS/MSRP *
a=setup:actpass a=connection:new a=path:msrps://ua.example.com:2855/
iau39;tcp a=accept-types: application/mcp+json a=sendrecv

## MSRP SEND carrying application/mcp+json

MSRP a786hjs2 SEND To-Path: msrps://bob.example.com:7394/iau39;tcp
From-Path: msrps://ua.example.com:2855/iau39;tcp Message-ID: 87652
Byte-Range: 1-172/172 Success-Report: yes Failure-Report: yes
Content-Type: application/mcp+json
```

```
   { "jsonrpc": "2.0", "id": 42, "method": "speech/event", "params":
   {"type":"vad_start","media":{"mid":"0","rtp_ts":367128000}} }
   -------a786hjs2$
```

Multimedia Tool Call with Binary Content (MCP over MSRP)

This example demonstrates a sophisticated multimedia tool call where an AI agent requests image analysis with the actual image data included in the MSRP message:

```
   MSRP img001 SEND To-Path: msrps://vision-agent.example.com:9000/
   abc123;tcp From-Path: msrps://client.example.com:9001/def456;tcp
   Message-ID: multimedia-tool-001 Byte-Range: 1-*/65536 Success-Report:
   yes Failure-Report: yes Content-Type: multipart/mixed; boundary="mcp-
   multimedia-boundary"

   --mcp-multimedia-boundary Content-Type: application/mcp+json

   { "jsonrpc": "2.0", "id": "img-analysis-001", "method": "tools/call",
   "params": { "name": "image_analysis", "arguments": { "image_ref":
   "cid:photo001", "analysis_type": "object_detection",
   "confidence_threshold": 0.8, "return_annotations": true } } }

   --mcp-multimedia-boundary Content-Type: image/jpeg Content-ID:
   <photo001> Content-Length: 65432

   [Binary JPEG data - 65,432 bytes] --mcp-multimedia-boundary--
   -------img001$
```

Streaming Tool Response (Large Document Generation)

This example shows how large tool responses can be streamed using MSRP chunking, enabling real-time processing of generated content:

```
   MSRP doc001 SEND To-Path: msrps://client.example.com:9001/def456;tcp
   From-Path: msrps://doc-agent.example.com:9002/ghi789;tcp Message-ID:
   streaming-response-001 Byte-Range: 1-4096/16384 Success-Report: no
   Failure-Report: yes Content-Type: application/mcp+json

   { "jsonrpc": "2.0", "id": "doc-gen-001", "result": { "type":
   "streaming_response", "chunk": 1, "total_chunks": 4, "content_type":
   "application/pdf", "data":
   "JVBERi0xLjQKMSAwIG9iago8PAovVHlwZSAvQ2F0YWxvZwovUGFnZXMgMiAwIFIK..."
   } } -------doc001$

   MSRP doc002 SEND To-Path: msrps://client.example.com:9001/def456;tcp
   From-Path: msrps://doc-agent.example.com:9002/ghi789;tcp Message-ID:
   streaming-response-002 Byte-Range: 4097-8192/16384 Success-Report: no
   Failure-Report: yes Content-Type: application/mcp+json
```

```
{ "jsonrpc": "2.0", "id": "doc-gen-001", "result": { "type":
"streaming_response", "chunk": 2, "total_chunks": 4, "content_type":
"application/pdf", "data":
"Pj4KZW5kb2JqCjIgMCBvYmoKPDwKL1R5cGUgL1BhZ2VzCi9LaWRzIFs..." } }
-------doc002$
```

Concurrent Tool Execution with Progress Reporting

This example demonstrates multiple concurrent tool calls with progress reporting, showcas
ing MSRP's ability to handle parallel operations:

```
MSRP batch001 SEND To-Path: msrps://processing-
agent.example.com:9003/jkl012;tcp From-Path:
msrps://client.example.com:9001/def456;tcp Message-ID: concurrent-
tools-001 Byte-Range: 1-256/256 Success-Report: yes Failure-Report:
yes Content-Type: application/mcp+json

{ "jsonrpc": "2.0", "id": "batch-process-001", "method": "tools/
batch_call", "params": { "tools": [ { "id": "task-A", "name":
"image_processing", "arguments": {"operation": "enhance",
"image_url": "..."} }, { "id": "task-B", "name": "text_analysis",
"arguments": {"text": "...", "analysis_type": "sentiment"} }, { "id":
"task-C", "name": "data_query", "arguments": {"query": "SELECT * FROM
users WHERE active=1"} } ] } } -------batch001$

MSRP progress001 SEND To-Path: msrps://client.example.com:9001/
def456;tcp From-Path: msrps://processing-agent.example.com:9003/
jkl012;tcp Message-ID: progress-update-001 Byte-Range: 1-128/128
Success-Report: no Failure-Report: yes Content-Type: application/
mcp+json

{ "jsonrpc": "2.0", "method": "tools/progress", "params": {
"batch_id": "batch-process-001", "completed": ["task-B"],
"in_progress": ["task-A", "task-C"], "progress": {"task-A": 0.6,
"task-C": 0.3} } } -------progress001$
```

Voice + Vision Integration with Temporal Correlation

This example shows the integration of audio streams with visual processing, demonstrating
 temporal correlation between RTP audio and MCP tool calls:

```
MSRP voice-vision001 SEND To-Path: msrps://multimodal-
agent.example.com:9004/mno345;tcp From-Path: msrps://voice-
client.example.com:9005/pqr678;tcp Message-ID: voice-vision-001 Byte-
Range: 1-512/512 Success-Report: yes Failure-Report: yes Content-
Type: application/mcp+json

{ "jsonrpc": "2.0", "id": "voice-vision-001", "method": "tools/call",
"params": { "name": "scene_analysis", "arguments": { "audio_context":
"User said: 'What do you see in this image?'", "image_ref":
```

```
"cid:camera-feed", "temporal_correlation": { "audio_mid": "0",
"rtp_timestamp": 367128000, "rtcp_ntp": "3923045130.125",
"speech_segment": { "start_time": "3923045128.500", "end_time":
"3923045130.125", "confidence": 0.95 } } } } } -------voice-
vision001$ '''
```

7. Security Considerations

This section provides comprehensive security analysis as required for IETF specifications. The combination of AI capabilities (MCP) with network signaling (SIP) creates unique security considerations that require careful analysis and mitigation.

7.1. Threat Model

7.1.1. Assets and Trust Boundaries

Protected Assets:

* AI agent capabilities and tool inventories
* MCP conversation data and context
* Authentication credentials and session state
* Business logic and decision-making processes
* Personal and organizational data processed by agents

Trust Boundaries:

* Network domain boundaries (inter-domain federation)
* Organizational boundaries (enterprise vs. external agents)
* Agent capability boundaries (tool access permissions)
* Session boundaries (dialog isolation)
* Transport boundaries (TLS termination points)

7.1.2. Threat Actors

External Attackers:

* Network-level attackers intercepting or modifying SIP traffic
* Malicious agents attempting to exploit other agents' capabilities

* Eavesdroppers seeking to extract sensitive AI conversation data
* Denial-of-service attackers targeting AI agent availability

Internal Threats:

* Compromised agents with legitimate network access
* Malicious insiders with SIP infrastructure access
* Misconfigured agents exposing excessive capabilities
* Rogue agents performing unauthorized tool execution

Infrastructure Threats:

* Compromised SIP proxies or registrars
* Man-in-the-middle attacks at TLS termination points
* DNS poisoning affecting agent discovery
* Certificate authority compromise

7.1.3. Attack Vectors

Capability Disclosure Attacks:

* Passive monitoring of MCP-Capabilities headers to map agent capabilities
* Registration-time capability enumeration via REGISTER inspection
* Feature-capability parameter harvesting from Contact headers
* OPTIONS method abuse to discover agent capabilities

Session Hijacking and Injection:

* SIP dialog hijacking to intercept MCP conversations
* Mid-dialog MESSAGE/INFO injection with malicious MCP payloads
* Session transfer attacks to redirect MCP conversations
* Re-INVITE attacks to modify MCP capability negotiations

Content and Protocol Attacks:

* Malformed MCP JSON-RPC payload injection
* Oversized payload attacks causing resource exhaustion
* MCP command injection through tool parameter manipulation
* Cross-protocol attacks leveraging SIP/MCP boundary confusion

Federation and Discovery Attacks:

* DNS poisoning to redirect agent discovery
* Rogue registrar attacks to capture agent registrations
* Inter-domain routing manipulation
* Certificate-based impersonation attacks

7.2. Security Requirements and Mitigations

7.2.1. Transport Security

Mandatory TLS Usage:

* All SIP signaling carrying MCP content MUST use TLS (SIPS)
* TLS version MUST be 1.2 or higher with forward secrecy
* Certificate validation MUST follow RFC 5922 (SIP TLS)
* MSRP sessions MUST use MSRPS (TLS-protected MSRP)
* WebSocket connections MUST use WSS (WebSocket Secure)

Certificate Management:

* Agents MUST validate peer certificates against trusted CAs
* Certificate pinning SHOULD be used for known agent relationships
* Certificate revocation checking MUST be implemented
* Mutual TLS authentication SHOULD be used for high-security deployments

7.2.2. Authentication and Authorization

Agent Authentication:

* SIP Digest authentication MUST be supported as baseline
* Certificate-based authentication SHOULD be preferred
* Multi-factor authentication MAY be required for sensitive agents
* Agent identity MUST be cryptographically bound to capabilities

Capability Authorization:

* MCP capabilities MUST be authorized per peer relationship
* Least-privilege principle MUST govern capability advertisement
* Dynamic capability restriction MUST be supported
* Tool execution MUST require explicit authorization

Session Authorization:

* Each MCP session MUST be independently authorized
* Session transfer MUST require re-authorization
* Capability escalation MUST trigger authorization checks
* Cross-domain sessions MUST respect federation policies

7.2.3. Content Protection

Payload Integrity:

* MCP payloads SHOULD use digital signatures for integrity
* S/MIME MAY be used for end-to-end payload protection
* JSON-RPC message IDs MUST be cryptographically secure
* Replay protection MUST be implemented using nonces/timestamps

Content Validation:

* All MCP payloads MUST be validated against JSON schema

* Tool parameters MUST be sanitized and validated
* Payload size limits MUST be enforced (recommend 1MB default)
* Malformed payloads MUST be rejected with appropriate SIP errors

Data Confidentiality:

* Sensitive MCP data SHOULD be encrypted end-to-end
* Capability information SHOULD be minimized in headers
* Logging MUST respect data classification and privacy requirements
* Memory handling MUST prevent sensitive data leakage

7.3. Feature Interaction Security Analysis

7.3.1. SIP-MCP Boundary Security

Protocol Confusion Attacks:

* Clear separation between SIP signaling and MCP application data
* MCP parsers MUST NOT interpret SIP headers as MCP content
* SIP parsers MUST treat MCP bodies as opaque application data
* Cross-protocol injection MUST be prevented through strict validation

Header Field Interactions:

* MCP-Capabilities and MCP-Select headers are informational only
* Header field values MUST NOT influence MCP protocol behavior
* Unknown header fields MUST be ignored per RFC 3261
* Header field size limits MUST be enforced

7.3.2. Multi-Modal Security Interactions

Audio-Data Correlation:

* RTP and MCP streams MUST maintain independent security contexts
* Temporal correlation MUST NOT leak sensitive information

* Audio content MUST NOT influence MCP tool execution
* MCP responses MUST NOT be automatically converted to audio

Session Transfer Security:

* MCP context MUST be securely transferred during SIP session mobility
* New endpoints MUST be re-authenticated before MCP continuation
* Capability re-negotiation MUST occur after session transfer
* Previous session state MUST be securely cleared

7.3.3. Federation Security Interactions

Inter-Domain Trust:

* Each domain MUST maintain independent MCP authorization policies
* Cross-domain capability sharing MUST be explicitly configured
* Federation agreements MUST specify MCP security requirements
* Domain boundaries MUST be enforced at the MCP application layer

Proxy Security:

* SIP proxies MUST NOT modify MCP-related content
* Proxy logs MUST NOT expose sensitive MCP capability information
* Route optimization MUST NOT bypass MCP security policies
* Proxy authentication MUST be independent of MCP authentication

7.4. Deployment-Specific Security Guidance

7.4.1. Enterprise Deployment

Network Security:

* Deploy SIP-aware firewalls with MCP content inspection
* Use network segmentation to isolate AI agent traffic
* Implement intrusion detection for abnormal MCP patterns

* Monitor capability advertisement for unauthorized disclosure

Policy Enforcement:

* Implement centralized MCP capability authorization
* Use SIP identity frameworks (RFC 8224) for agent authentication
* Deploy policy servers for dynamic capability control
* Audit all MCP tool executions and results

Operational Security:

* Regular security assessment of agent capabilities
* Incident response procedures for compromised agents
* Secure agent provisioning and deprovisioning
* Staff training on AI-specific security risks

7.4.2. Federated Deployment

Inter-Organization Security:

* Establish formal security agreements for MCP federation
* Use mutual TLS with organization-specific certificate authorities
* Implement capability filtering at domain boundaries
* Monitor cross-domain MCP traffic for anomalies

Trust Management:

* Maintain explicit trust relationships between organizations
* Regular review and update of federated agent permissions
* Implement capability revocation mechanisms
* Cross-organization incident response coordination

7.4.3. Cloud and Service Provider Deployment

Multi-Tenancy Security:

* Strict isolation between different customer agents
* Tenant-specific capability authorization policies
* Encrypted storage of MCP conversation data
* Audit trails for all cross-tenant interactions

Service Provider Responsibilities:

* Secure agent hosting and capability management
* Regular security updates and vulnerability management
* Customer data protection and privacy compliance
* Transparent security incident reporting

7.5. Privacy Considerations

7.5.1. Data Minimization

Capability Advertisement:

* Advertise only necessary capabilities for intended interactions
* Use capability filtering based on peer identity and context
* Implement dynamic capability advertisement based on session needs
* Regular review and pruning of advertised capabilities

Conversation Data:

* Minimize retention of MCP conversation logs
* Implement data classification for different types of MCP content
* Use data anonymization techniques where appropriate
* Respect user consent for AI conversation data processing

7.5.2. Regulatory Compliance

GDPR and Similar Regulations:

* Implement data subject rights for MCP conversation data

* Provide clear notice about AI agent data processing
* Support data portability for MCP conversation exports
* Implement right to erasure for MCP-related data

Industry-Specific Requirements:

* Healthcare: HIPAA compliance for medical AI agents
* Finance: PCI DSS compliance for payment-related agents
* Government: Appropriate security clearance levels for classified agents
* Legal: Attorney-client privilege protection for legal AI agents

7.6. Security Monitoring and Incident Response

7.6.1. Monitoring Requirements

Real-Time Monitoring:

* Anomalous MCP capability advertisement patterns
* Unusual tool execution frequencies or patterns
* Failed authentication attempts for agent access
* Suspicious cross-domain MCP traffic patterns

Audit Requirements:

* Complete audit trail of all MCP tool executions
* Agent capability changes and authorization updates
* Session establishment and termination events
* Security policy violations and enforcement actions

7.6.2. Incident Response

Detection and Classification:

* Automated detection of MCP-specific security events
* Classification of incidents by severity and impact

* Integration with existing security incident response procedures
* Specialized procedures for AI agent compromise

Response and Recovery:

* Immediate capability revocation for compromised agents
* Session termination and cleanup procedures
* Evidence preservation for MCP-related security incidents
* Communication procedures for federated incident response

7.7. Implementation Security Guidelines

7.7.1. Secure Development Practices

Code Security:

* Input validation for all MCP content parsing
* Secure memory management for sensitive MCP data
* Regular security code reviews focusing on SIP-MCP interactions
* Automated security testing for MCP protocol implementations

Cryptographic Implementation:

* Use well-established cryptographic libraries
* Proper random number generation for MCP session identifiers
* Secure key management for MCP-related cryptographic operations
* Regular cryptographic algorithm updates and security patches

7.7.2. Configuration Security

Secure Defaults:

* Minimal capability advertisement by default
* Strict authentication requirements by default
* Conservative timeout and rate limiting settings

* Comprehensive logging enabled by default

Configuration Management:

* Secure storage of agent configuration data

* Version control and audit trails for configuration changes

* Automated configuration validation and security checking

* Regular security configuration reviews and updates

This comprehensive security analysis addresses the unique risks introduced by combining AI capabilities with SIP signaling, providing specific guidance for secure deployment and operation of MCP-over-SIP systems.

8. IANA Considerations

This document requests IANA registration of SIP protocol elements as described below. As an Informational RFC, these registrations follow the Designated Expert review process per RFC 5727.

8.1. Registration of Option-Tag

Per RFC 5727, SIP option tags require Standards Action for registration. This Informational specification does not request registration of the "mcp" option tag. Implementations using this specification SHOULD use an experimental or private option tag (e.g., "x-mcp" or organization-specific variants) until a Standards Track specification is available.

Note for Future Standards Track Work: Name: *mcp* Description: Support for SIP MCP extension Reference: [Future Standards Track RFC]

8.2. Registration of Header Fields

The following header fields are requested for registration under the Designated Expert review process per RFC 5727:

* *Header Field Name:* MCP-Capabilities

* *Compact Form:* none

* *Reference:* This document

* *Registration Type:* Informational (Designated Expert Review)

* *Header Field Name:* MCP-Select

* *Compact Form:* none

* *Reference:* This document

* *Registration Type:* Informational (Designated Expert Review)

8.3. Registration of Feature-Capability Indicators (RFC 6809)

The following feature-capability indicators are requested for registration:

* *Indicator:* +mcp

* *Reference:* This document

* *Registration Type:* Informational (Designated Expert Review)

* *Indicator:* +mcp.ver

* *Reference:* This document

* *Registration Type:* Informational (Designated Expert Review)

* *Indicator:* +mcp.cap

* *Reference:* This document

* *Registration Type:* Informational (Designated Expert Review)

8.4. Media Type Registration

This document requests registration of the following media type:

Type name: application *Subtype name:* mcp+json *Required parameters:* none *Optional parameters:* charset (defaults to UTF-8) *Encoding considerations:* binary; typically UTF-8 JSON *Security considerations:* see Section 10 *Interoperability considerations:* none *Published specification:* This document *Applications that use this media type:* SIP UAs implementing MCP extension *Fragment identifier considerations:* n/a *Additional information:* n/a *Person & email to contact for further information:* [Author contact information] *Intended usage:* LIMITED USE (see Applicability Statement in Section 3.1) *Restrictions on usage:* See Section 3.1 for deployment limitations *Author:* Thomas McCarthy-Howe *Change controller:* IETF

8.5. Designated Expert Considerations

Per RFC 5727, the Designated Expert reviewing registrations from this document should verify:

1. The proposed registrations do not conflict with existing SIP protocol elements
2. The security considerations have been adequately addressed
3. The applicability statement clearly defines appropriate usage scenarios
4. The registrations follow established SIP extension patterns and do not undermine SIP’s architectural integrity

9. References

9.1. Normative

* *[RFC2119]* Bradner, S., "Key words for use in RFCs...", BCP 14.
* *[RFC3261]* Rosenberg, J., et al., "SIP: Session Initiation Protocol".
* *[RFC3264]* Rosenberg, J., Schulzrinne, H., et al., "An Offer/Answer Model...".
* *[RFC5234]* Crocker, D., Overell, P., "Augmented BNF for Syntax".
* *[RFC3711]* Baugher, M., et al., "The Secure Real-time Transport Protocol (SRTP)".
* *[RFC4145]* Yon, D., et al., "TCP-Based Media Transport in the SDP (comedia)".
* *[RFC4975]* Campbell, B., et al., "The Message Session Relay Protocol (MSRP)".
* *[RFC4976]* Mahy, R., et al., "MSRP Relays for NAT Traversal".
* *[RFC5764]* McGrew, D., Rescorla, E., "DTLS-SRTP".
* *[RFC8866]* Begen, A., et al., "Session Description Protocol (SDP)".
* *[RFC3550]* Schulzrinne, H., et al., "RTP: A Transport Protocol for Real-Time Applications".

9.2. Informative

* *[RFC7118]* Baz Castillo, I., et al., "The WebSocket Protocol as a SIP Transport".

9.3. A. Acknowledgments

Thanks to the SIP and ART area reviewers for early feedback.

9.4. B. Change Log

* *-00* Initial version; added Section 2 introducing MCP; added Section 7.5 on multimodal operation and Examples 9.4-9.5; added Section 4.1 on agent-to-agent interoperation with two use cases.

10. References

10.1. 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>.

[RFC3261] Rosenberg, J., Schulzrinne, H., Camarillo, G., Johnston, A., Peterson, J., Sparks, R., Handley, M., and E. Schooler, "SIP: Session Initiation Protocol", RFC 3261, DOI 10.17487/RFC3261, June 2002, <https://www.rfc-editor.org/rfc/rfc3261>.

[RFC3264] Rosenberg, J. and H. Schulzrinne, "An Offer/Answer Model with Session Description Protocol (SDP)", RFC 3264, DOI 10.17487/RFC3264, June 2002, <https://www.rfc-editor.org/rfc/rfc3264>.

[RFC3550] Schulzrinne, H., Casner, S., Frederick, R., and V. Jacobson, "RTP: A Transport Protocol for Real-Time Applications", STD 64, RFC 3550, DOI 10.17487/RFC3550, July 2003, <https://www.rfc-editor.org/rfc/rfc3550>.

[RFC3711] Baugher, M., McGrew, D., Naslund, M., Carrara, E., and K. Norrman, "The Secure Real-time Transport Protocol (SRTP)", RFC 3711, DOI 10.17487/RFC3711, March 2004, <https://www.rfc-editor.org/rfc/rfc3711>.

[RFC4145] Yon, D. and G. Camarillo, "TCP-Based Media Transport in the Session Description Protocol (SDP)", RFC 4145, DOI 10.17487/RFC4145, September 2005, <https://www.rfc-editor.org/rfc/rfc4145>.

[RFC4975] Campbell, B., Ed., Mahy, R., Ed., and C. Jennings, Ed., "The Message Session Relay Protocol (MSRP)", RFC 4975, DOI 10.17487/RFC4975, September 2007, <https://www.rfc-editor.org/rfc/rfc4975>.

[RFC4976] Jennings, C., Mahy, R., and A. B. Roach, "Relay Extensions for the Message Sessions Relay Protocol (MSRP)", RFC 4976, DOI 10.17487/RFC4976, September 2007, <https://www.rfc-editor.org/rfc/rfc4976>.

[RFC5234] Crocker, D., Ed. and P. Overell, "Augmented BNF for Syntax Specifications: ABNF", STD 68, RFC 5234, DOI 10.17487/RFC5234, January 2008, <https://www.rfc-editor.org/rfc/rfc5234>.

[RFC5727] Peterson, J., Jennings, C., and R. Sparks, "Change Process for the Session Initiation Protocol (SIP) and the Real-time Applications and Infrastructure Area", BCP 67, RFC 5727, DOI 10.17487/RFC5727, March 2010, <https://www.rfc-editor.org/rfc/rfc5727>.

[RFC5764] McGrew, D. and E. Rescorla, "Datagram Transport Layer Security (DTLS) Extension to Establish Keys for the Secure Real-time Transport Protocol (SRTP)", RFC 5764, DOI 10.17487/RFC5764, May 2010, <https://www.rfc-editor.org/rfc/rfc5764>.

[RFC6809] Holmberg, C., Sedlacek, I., and H. Kaplan, "Mechanism to Indicate Support of Features and Capabilities in the Session Initiation Protocol (SIP)", RFC 6809, DOI 10.17487/RFC6809, November 2012, <https://www.rfc-editor.org/rfc/rfc6809>.

[RFC7587] Spittka, J., Vos, K., and JM. Valin, "RTP Payload Format for the Opus Speech and Audio Codec", RFC 7587, DOI 10.17487/RFC7587, June 2015, <https://www.rfc-editor.org/rfc/rfc7587>.

[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>.

[RFC8866] Begen, A., Kyzivat, P., Perkins, C., and M. Handley, "SDP: Session Description Protocol", RFC 8866, DOI 10.17487/RFC8866, January 2021, <https://www.rfc-editor.org/rfc/rfc8866>.

10.2. Informative References

[RFC7118] Baz Castillo, I., Millan Villegas, J., and V. Pascual, "The WebSocket Protocol as a Transport for the Session Initiation Protocol (SIP)", RFC 7118, DOI 10.17487/RFC7118, January 2014, <https://www.rfc-editor.org/rfc/rfc7118>.

Author’s Address

Thomas McCarthy-Howe
VCONIC
Email: ghostofbasho@gmail.com