MCP의 SDK 간 프로토콜 적합성 분석

김아인[1], 이승수[2]
인천대학교 (학부생[1], 교수[2])

Cross-SDK Protocol Conformance Analysis of MCP

Ein Kim[1], Seungsoo Lee[2],
Incheon National University (Undergraduate student[1], Professor[2])

요 약

MCP(Model Context Protocol)는 LLM과 외부 자원을 연결하는 표준으로, 현재 10개 프로그래밍 언어의 공식 SDK가 제공되고 있으나 이들 구현의 프로토콜 명세 준수 여부에 대한 체계적 검증은 부재하다. 본 연구는 이 공백을 해소하고자 최초의 교차 SDK 프로토콜 적합성 측정을 수행한다. 7개 공식 SDK의 14개 SDK-전송(stdio, Streamable-HTTP) 조합을 대상으로 서버 비정상 종료 · 세션 상태 변조 · 응답 명세 준수 여부의 세 가지 범주를 검증하고자 다단계 테스트 하네스를 구축하고 재현성 통계를 수집했다. 실험 결과 7개 SDK 중 6종에서 하나 이상의 적합성 결함이 확인되었으며, 모든 범주에서 stdio가 Streamable-HTTP 대비 일관되게 낮은 적합성을 보이는 전송 이분성(transport dichotomy)이 관찰되었다. 모든 결과는 책임 공개(responsible disclosure) 절차에 따라 각 SDK 관리자에게 취약점 보고 및 버그 리포트로 전달되었다.

I. Introduction

MCP는 LLM과 외부 자원 간 구조화된 통신을 지원하는 JSON-RPC 2.0 기반 표준이다. 2024년 11월 공개 이후 파일시스템·데이터베이스 접근, 웹 브라우징 등 다양한 기능을 제공하는 서버와 Claude, ChatGPT 등 호스트를 포함하는 생태계가 급격히 확장되었으며, 현재 10개 언어의 공식 SDK가 제공되고 있다.

이처럼 신뢰 경계를 형성하는 프로토콜에서 명세 준수는 필수적이다. 적합성 결함이 있는 MCP 서버는 협상된 세션 상태를 사후 변경하거나 유효한 요청에 무응답하여 AI 에이전트 워크플로의 신뢰성을 저해할 수 있다. 또한 명세가 규정하지 않는 악의적 입력에 따른 비정상 종료는 가용성 저하를 초래한다. 그럼에도 불구하고 프로토콜 적합성에 대한 체계적인 검증은 이루어지지 않았다.

본 논문의 기여는 다음과 같다. (1) 명세의 규범적 요구사항으로부터 도출한 세 가지 범주에 대한 이중 전송 테스트 하네스를 설계한다. (2) 14개 SDK-전송 조합에 대해 42개 테스트 캠페인을 실행하고 재현성 통계를 수집한다. (3) 세 범주 모두에서 stdio가 HTTP에 대비 일관되게 적합성이 낮은 전송 이분성을 확인한다. (4) 전 결과를 SDK 유지관리자에게 취약점 및 버그 리포트로 보고하고 명세 개선안을 제안한다.

II. Background

2.1. MCP 프로토콜 및 명세

MCP 연결은 초기화-운영-종료의 세 단계로 구성된다. 초기화 단계에서 클라이언트와 서버는 기능과 프로토콜 버전을 협상하며, 이후 운영 단계에서 양측은 협상된 조건만을 사용하여 메시지를 교환한다. MCP 명세는 RFC 2119(MUST, MUST NOT, SHOULD, MAY)로 요구사항의 강도를 구분하며, 이를 세 종류로 나누어 각각 §3의 적합성 기준에 대응한다.

(1) 서버 동작을 직접 규정하는 명시적 의무로, 'id가 포함된 요청에는 반드시 결과 또는 오류 응

Table 1: Systems Under Test: SDK Versions and Reference Servers

SDK	Package	Version	stdio Server	HTTP Server
TypeScript	`@modelcontextprotocol/sdk`	≤ 1.27.1	`server-everything`	`server-everything`
Python	`mcp` (pip)	≤ 1.26.0	`mcp-server-fetch`	`everything-server`
Go	`go-sdk` (Go)	≤ 1.4.1	`everything`	`everything`
Rust	`rmcp` (Rust)	≤ 1.2.0	`counter_stdio`	`counter_streamhttp`
Ruby	`mcp` (RubyGems)	≤ 0.9.0	`stdio_server`	`http_server`
Kotlin	`kotlin-sdk` (Maven)	≤ 0.9.0	`kotlin-mcp-server`	`simple-streamable`
PHP	`mcp/sdk` (Packagist)	≤ 0.4.0	`discovery-calculator`	`discovery-calculator`

답을 반환해야 한다'[2] 등이 이에 해당한다.

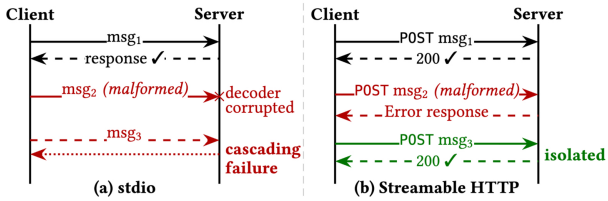
(2) 서버 대응 동작이 정의되지 않은 요구사항이다. '첫 번째 상호작용은 반드시 초기화 요청이어야 한다'[1]를 명시하면서도, 운영 중 초기화 요청 수신 시 서버의 처리 방식은 규정하지 않는다.

(3) 명세에 지침이 없는 공백 영역으로, JSON 파싱의 최대 중첩 깊이가 규정되지 않아 깊이 중첩된 입력에 대한 서버 동작이 정의되지 않는다.

2.2. 전송 아키텍처

MCP는 두 가지 전송을 정의한다(Figure 1).

Figure 1: Example failure behavior across transports.



(1) stdio. 클라이언트는 서버를 서브프로세스로 생성하여 표준 입출력 바이트 스트림 위에서 단일행 JSON-RPC 메시지를 교환하며, 세션 수명은 프로세스의 수명에 종속된다. 모든 메시지가 하나의 스트림 디코더 상태를 공유하므로, 파싱 오류가 후속 메시지 처리에 영향을 줄 수 있다.

(2) Streamable-HTTP. 클라이언트 메시지는 독립적인 POST로, 서버 주도 메시지는 POST 응답 또는 별도 GET 스트림 기반의 SSE로 전달될 수 있다. 서버가 초기화 시 MCP-Session-Id 헤더를 할당하면 클라이언트는 이후 모든 요청에 해당 ID와 MCP-Protocol-Version 헤더를 포함해야 한다. 각 POST 요청이 독립적으로 처리되므로 파싱 오류는 다른 요청에 전파되지 않는다.

III. Analysis and Findings

3.1. 적합성 기준 및 실험 설계

두 전송층을 동일한 인터페이스로 추상화하는 테스트 하네스를 구축하고, 각 이슈 클래스에 대해 유효한 초기화 후 다단계 시퀀스를 10회 반복 실행하여 재현성을 수집한다. MCP와 JSON-RPC 2.0 명세로부터 세 가지 적합성 기준을 도출했다.

modelcontextprotocol 조직[3]의 servers 저장소 또는 각 SDK 저장소가 제공하는 공식 레퍼런스 서버를 사용했으며, 16개의 SDK-전송 조합과 42개의 테스트 캠페인을 구성했다(Table 1).

심층 중첩. §2.1의 (3)에서 확인한 공백 영역에 해당한다. 서버가 처음 실패하는 깊이, 페이로드 크기, 서버 프로세스 생존 여부를 기록한다.

중복 초기화. §2.1의 (2)에 따라 기존 세션 상태를 덮어쓰는 재초기화 동작 여부를 검증한다.

무응답. §2.1의 (1)에 따라, 초기화 이후 id가 포함된 메시지를 전송하여 응답 의무를 확인한다.

3.2. 심층 중첩에 의한 자원 고갈

TS, Kotlin, PHP를 제외한 모든 SDK가 stdio에서 심층 중첩 JSON 입력에 대해 비정상 종료 및 타임아웃을 보였다. 특히 Rust는 1,218바이트의 소량 페이로드만으로 서버 프로세스가 종료되어 공격 비용이 극히 낮다. 임계 중첩 깊이는 SDK별로 최대 100배 편차를 나타냈다.

- Rust·Python은 재귀 한도가 각각 128단계와 256단계로 낮아, 한도 초과 시 예외가 프로세스로 전파되며 서버가 강제 종료된다.
- Go는 encoding/json에 하드코딩된 10,000 단계를 초과하면 스트림 디코더 상태 손상이 전송 계층 해제로 이어지는 연쇄 장애가 발생한다.
- Ruby는 256단계 이전에 오류를 즉시 반환하고, 이후에는 7초 타임아웃 발생 후 자동 회복한다.
- TypeScript의 V8엔진은 10만 단계(1.08 MB)의 페이로드를 163 ms 만에 유연하게 처리했다.
- PHP와 Kotlin은 최대 10,000단계에서도 적정 오류 코드를 반환하며 프로세스 가용성을 유지했다.
- HTTP 전송 계층에서는 영향이 없었다. Go의는 10,000 깊이에서도 정상 동작하였고, Rust는 파싱 오류 응답 후 서버 프로세스를 유지하였다.

3.3. 중복 초기화에 의한 세션 상태 변조

7개 SDK 중 6종이 중복 초기화를 수락했으며, 8가지 변형 테스트 및 10회 반복 실험 결과 재현율 100%로 수락·거부 양상이 결정적이었다.

• 협상이 완료된 버전 및 기능 명세가 사후 변경 가능함을 실증했다. 이는 단일 세션 내에서 프로토콜 수준을 변경하거나 LLM 및 사용자 상호작용 권한이 변경될 수 있음을 의미한다.

• PHP가 유일하게 양쪽 전송 계층에서 세션 기반 검증으로 거부 메시지를 반환하여 생명주기 제약을 유일하게 강제했으며, Go·Rust는 전송 계층에 무관하게 중복 초기화를 수락했다.

• TypeScript, Python, Ruby, Kotlin 4개 SDK는 stdio에서는 수락하면서 HTTP에서는 거부하여, 전송 계층 간 검증 비대칭이 나타났다.

3.4. 응답 의무 미준수

7개 stdio 구현 중 6개에서 메시지 폐기가 관찰된 반면, HTTP 구현은 모두 정상 응답하였다. 이런 폐기 현상은 10회 반복 실험 시 100% 재현율로 완전히 결정적이었다(Table 2).

Table 2: Per-Message Silent Drop Matrix in STDIO

Tier	Test Message	TS	Py	Go	Rust	Ruby	Kotlin	PHP
T1	Wrong `jsonrpc` version	✗	✗	✗	✗	✗	–	✗
T1	Missing `jsonrpc` field	✗	✗	✗	✗	✗	–	✗
T1	Method as number	✗	✗	✗	✗	✗	–	✗
T1	Null params	✗	✗	✗	✗	–	–	–
T1	Missing required params	–	–	–	–	–	–	–
T1	Extra top-level field	✗	–	–	–	–	–	–
T1	`__proto__` injection	–	✗	✗	✗	–	–	–
T2	Params as array	✗	✗	✗	✗	–	–	–
T2	Id as null	–	–	–	–	–	–	–
T3	Empty method string	–	✗	✗	✗	–	–	✗
T3	Non-existent method	–	✗	✗	✗	–	–	✗
T3	Id as string	–	✗	✗	✗	–	–	–

✗ = silent drop; – = proper error response; T1 = structurally malformed. T2 = MCP-specific restriction; T3 = semantically valid input.

• T1의 공통 원인은 구조적으로 잘못된 메시지가 stdio 계층의 느슨한 디코딩으로 유입된 뒤, 적절한 에러 핸들링 없이 무응답으로 처리되거나 알림으로 오분류 되는 견고성 결함이다.

• Go·Python·Rust SDK는 규격상 유효한 문자열 타입 ID를 가진 요청(T3)을 폐기하는 명세 준수 결함을 보인다. 이는 클라이언트의 무한 대기를 유발하여 서비스 거부 조건을 형성할 수 있다.

• Kotlin SDK는 유일하게 stdio 상의 모든 메시지에 대해 적절한 오류 코드를 반환하였다.

3.5. 전송 계층 이분성

모든 결함 범주에서 stdio 구현이 HTTP 대비 낮은 적합성을 보이며 전송층 간 명확한 이분성을 드러냈다. 이 격차는 두 가지 구조적 요인으로 설명할 수 있다. 첫째, 앞서 기술한 전송 구조 차이로 HTTP에서는 파싱 오류가 다른 요청에 전파되지 않는 반면, stdio에서는 디코더 상태 손상이 후속 메시지 처리에 영향을 줄 수 있다. 둘째, 다수 HTTP 구현이 Mcp-Session-Id 헤더 확인과 같은 세션 수준 검증을 메시지 처리에 앞서서 수행하여, 잘못된 동작을 전송층 경계에서 차단한다.

IV. Conclusion

본 연구는 7개 공식 MCP SDK 의 14개 조합을 대상으로 최초의 체계적 프로토콜 적합성 측정을 수행했다. 분석 결과 심층 중첩 시 비정상 종료, 중복 초기화에 따른 세션 상태 변조, 응답 명세 미준수의 3대 결함 유형을 확인했으며, 모든 유형에서 stdio의 준수도가 낮은 전송층 이분성이 일관되게 관찰되었다. 특히 중복 초기화 실험 결과는 명세에 보완이 필요한 지점을 시사한다. 서버가 중복 초기화를 거부해야 한다는 명시적 조항이 없어 해석상 차이가 발생할 수 있으므로, 이를 해소하기 위해 서버 측 의무를 보다 명확히 규정할 필요가 있다. 개선 방안으로 ①중복 초기화 거부 의무 조항 명시 ②stdio 계층의 메시지 검증 강화를 제안한다. 본 연구를 통해 발견된 모든 결함은 책임 있는 공개 절차에 따라 각 SDK 관리자에게 취약점 및 버그 리포트로 보고되었다. 향후에는 적합성 및 취약성 테스트를 자동화하는 프로토콜 침투 테스트 프레임워크를 연구할 예정이다.

Acknowledgements

이 성과는 정부(과학기술정보통신부)의 재원으로 한국연구재단의 지원을 받아 수행된 연구임 (No. RS-2025-16069415).

[참고문헌]

[1] Anthropic, “Model Context Protocol Specification”, https://modelcontextprotocol.io/specification/2025-11-25

[2] JSON-RPC Working Group, “JSON-RPC 2.0 Specification”, https://www.jsonrpc.org/specification

[3] Model Context Protocol, “modelcontextprotocol GitHub organization”, https://github.com/modelcontextprotocol