**Reporte de Implementación MCP**

CC3067 Redes — Proyecto 1  
Uso de un protocolo existente

*Autor: Gabriel Alberto Paz González  
Fecha: 25 de septiembre de 2025*

Resumen

Este documento describe la implementación y evaluación de múltiples servidores MCP (Model Context Protocol) integrados en un host tipo chatbot. Se documentan las especificaciones, parámetros y endpoints de los servidores MCP desarrollados por el autor (SiteLens y RemoteMCP), así como el análisis de la comunicación remota capturada con Wireshark a nivel de las capas de enlace, red, transporte y aplicación. Finalmente, se presentan conclusiones y comentarios sobre el proyecto.

# Especificación de los servidores MCP desarrollados

## SiteLens (MCP local por STDIO)

Descripción: Servidor MCP local (Node/TypeScript) que audita carpetas con HTML estático sin red.

Transporte: STDIO.

Protocolo MCP: 2024-11-05; métodos soportados: initialize, tools/list, tools/call.

Control de acceso: whitelist de raíces vía --roots o variable ALLOWED\_ROOTS (separadas por ';' en Windows).

Formato de respuesta tabulable: { "structuredContent": { "result": <lista|obj> } }.

* Tools expuestas:
* aa.allowed\_roots → {} → string[] de roots efectivas.
* aa.sitemap → { path, includeHtmlOnly?, maxDepth? } → árbol JSON del FS.
* aa.link\_check → { path, entry?, extensions? } → validación de enlaces internos (externos: skipped).
* aa.asset\_budget → { path, patterns?, budgetKB? } → totales por tipo, top pesados y sobre presupuesto.
* aa.scan\_accessibility → { path, include?, exclude? } → reglas WCAG-lite: alt, labels, landmarks, headings, contraste.
* aa.report → { path, weights?, top? } → ranking (0–100) y quick wins consolidando resultados previos.

Ejemplo de integración en el host (mcp\_config.json):

{  
 "name": "SiteLens",  
 "transport": "stdio",  
 "command": "node",  
 "args": ["C:/UVG/PROYECTO1/MCP\_Local/sitelens/dist/server.js", "--roots", "C:/UVG/PROYECTO1/MCP\_Local/test/site"]  
}

## RemoteMCP (MCP remoto vía Cloudflare Workers)

Descripción: Servidor MCP remoto (Cloudflare Workers) con endpoints WSS y HTTP JSON-RPC 2.0.

Transporte: WebSocket seguro (WSS) y HTTP (POST).

Ruta de servicio: /mcp

Protocolo MCP: métodos initialize, tools/list, tools/call.

Formato de respuesta tabulable: { "structuredContent": { "result": <obj> } }.

* Endpoints:
* WSS: wss://remote-mcp-demo.<subdominio>.workers.dev/mcp
* HTTP: https://remote-mcp-demo.<subdominio>.workers.dev/mcp
* Tools expuestas (demo):
* remote.ping → {} → { ok:true, message:"pong" }
* remote.time → {} → { now:"<ISO>" }
* remote.echo → { text:string } → { echo:"<text>" }

Ejemplo de integración en el host (mcp\_config.json):

{  
 "name": "RemoteMCP",  
 "transport": "websocket",  
 "url": "wss://remote-mcp-demo.<subdominio>.workers.dev/mcp"  
}

# Análisis por capas (basado en captura Wireshark del punto 8)

A continuación se explica el flujo típico de una interacción entre el host y RemoteMCP, desglosado por capa del modelo OSI/TCP-IP:

## Capa de enlace (Link)

La transmisión se realiza sobre un medio físico/lógico (Ethernet o Wi‑Fi). Se observan tramas con direcciones MAC origen/destino y control de errores (FCS). La segmentación en tramas y el acceso al medio (CSMA/CD o CSMA/CA) son transparentes a las capas superiores.

## Capa de red (IP)

Sobre la capa de enlace, IP enruta paquetes entre el host y los servidores de Cloudflare. Se aprecian direcciones IP origen/destino públicas, TTL decreciente en tránsito, y posible fragmentación si fuera necesario. La resolución de nombres (DNS) precede a la conexión (lookup de remote-mcp-demo.<subdominio>.workers.dev).

## Capa de transporte (TCP/TLS)

El canal usa TCP (puerto 443). Se observa el three-way handshake (SYN, SYN/ACK, ACK), seguido por el establecimiento de la sesión TLS (ClientHello/ServerHello, intercambio de claves). Para descifrar el contenido en Wireshark se utilizó SSLKEYLOGFILE, permitiendo ver el tráfico WSS/HTTPS en claro.

## Capa de aplicación (HTTP/WS + JSON‑RPC + MCP)

Una vez establecido TLS, la aplicación utiliza HTTP/1.1 o HTTP/2 con upgrade a WebSocket (en WSS) o POST (en HTTP). El payload transporta mensajes JSON‑RPC 2.0 con los métodos del protocolo MCP.

Clasificación de mensajes observados:

* - Sincronización: `initialize` (request) y su `result` (response).
* - Descubrimiento: `tools/list` (request) y su `result` (response) con metadatos de cada tool.
* - Ejecución: `tools/call` (request) con `{ name, arguments }` y su `result`/`error` (response).

Ejemplo de request/response (JSON‑RPC):

{ "jsonrpc":"2.0", "id":1, "method":"initialize" }  
{ "jsonrpc":"2.0", "id":1, "result": { "serverInfo": { "name":"RemoteMCP-Demo","version":"1.0.0" }, "capabilities":{ "tools":{} } } }

# Conclusiones y comentarios

El protocolo MCP permitió integrar de forma homogénea servidores locales y remotos, desacoplando la lógica de las herramientas del LLM/host. SiteLens demostró un caso no trivial de auditoría de HTML estático con reglas WCAG-lite, mientras que RemoteMCP evidenció la factibilidad de exponer herramientas por WSS/HTTP en la nube con baja latencia. La captura y análisis con Wireshark confirmó la secuencia de sincronización y ejecución en JSON‑RPC sobre TLS, y ayudó a comprender la interacción de las capas. Entre los retos encontrados estuvieron el alineamiento de tipados (TypeScript vs. runtime de Workers), la normalización segura de rutas (whitelist de raíces) y el manejo explícito de schemas de entrada/salida para garantizar respuestas MCP válidas. En conjunto, el proyecto cumple los objetivos de comprender e implementar MCP en escenarios locales y remotos, integrándolo en un host basado en la API de OpenAI.

# Referencias

1. Anthropic. (2025). Model Context Protocol Specification (2025‑06‑18). https://modelcontextprotocol.io/specification/2025-06-18
2. JSON‑RPC Working Group. (n.d.). JSON‑RPC 2.0 Specification. https://www.jsonrpc.org/specification
3. Model Context Protocol. (n.d.). Learn — Architecture. https://modelcontextprotocol.io/docs/learn/architecture
4. Cloudflare. (n.d.). Cloudflare Workers Documentation. https://developers.cloudflare.com/workers/
5. Universidad del Valle de Guatemala. (2025). Proyecto 1: Uso de un protocolo existente (Instrucciones de curso).

# Apéndice A — Configuración del host (mcp\_config.json)

{  
 "servers": [  
 {  
 "name": "SQLScout",  
 "transport": "stdio",  
 "command": "python",  
 "args": ["-B", "-m", "src.server\_mcp"],  
 "cwd": "C:/UVG/PROYECTO1/MCP\_Local/SQL\_MCP",  
 "env": {}  
 },  
 {  
 "name": "FS",  
 "transport": "stdio",  
 "command": "C:/Program Files/nodejs/npx.cmd",  
 "args": ["-y","@modelcontextprotocol/server-filesystem","C:/UVG/PROYECTO1/MCP\_Local"],  
 "cwd": ".",  
 "env": {}  
 },  
 {  
 "name": "Git",  
 "transport": "stdio",  
 "command": "py",  
 "args": ["-3.11","-m","mcp\_server\_git","--repository","C:/UVG/PROYECTO1/MCP\_Local"],  
 "cwd": "."  
 },  
 {  
 "name": "SiteLens",  
 "transport": "stdio",  
 "command": "node",  
 "args": ["C:/UVG/PROYECTO1/MCP\_Local/sitelens/dist/server.js","--roots","C:/UVG/PROYECTO1/MCP\_Local/test/site"]  
 },  
 {  
 "name": "anime-helper",  
 "transport": "stdio",  
 "command": "py",  
 "args": ["-3.11","-m","anime\_helper.server"],  
 "cwd": ".",  
 "env": {}  
 },  
 {  
 "name": "RemoteMCP",  
 "transport": "http",  
 "url": "https://remote-mcp-demo.gabouvg.workers.dev/mcp"  
 }  
 ]  
}