**Informe Final – Proyecto 1: Uso de un Protocolo Existente (MCP-Schedulizer)**

**Curso:** CC3067 – Redes de Computadoras  
**Nombre del Proyecto:** MCP-Schedulizer  
**Estudiante:** Brandon Javier Reyes Morales  
**Carné:** 22992  
**Fecha:** Septiembre 2025

**1. Introducción**

En este proyecto se implementó un sistema de integración de protocolos existente bajo el estándar **Model Context Protocol (MCP)**. El objetivo fue crear un cliente anfitrión (chatbot) que se comunique con servidores MCP para generar, listar y manipular agendas semanales basadas en tareas. Se implementaron múltiples servidores (local, remoto en Google Cloud Run, y filesystem oficial) para evaluar diferentes medios de comunicación (HTTP y STDIO) usando JSON-RPC 2.0.

**2. Objetivo General**

Desarrollar un sistema cliente-servidor funcional, basado en el protocolo MCP, que permita al usuario interactuar con diferentes servicios relacionados a tareas, agendas y contenido educativo, mediante llamadas remotas a servidores locales y remotos.

**3. Herramientas Utilizadas**

* **Lenguaje de Programación:** Python 3.12
* **Librerías:** requests, anthropic, mcp, mcp-server-openapi, python-dotenv
* **Servidor Remoto:** Google Cloud Run (https://mcp-schedulizer-v1-abcdef.a.run.app)
* **Servidor Oficial:** Filesystem server vía STDIO (npx @modelcontextprotocol/server-filesystem)
* **Cliente Chatbot:** cliente\_chatbot.py
* **Herramientas de red:** Wireshark, nslookup, netstat

**4. Descripción de Arquitectura del Proyecto**

El proyecto se compone de los siguientes módulos:

**4.1 Cliente Chatbot (cliente\_chatbot.py)**

Un asistente que interactúa con el usuario, procesando comandos para:

* Inicializar servidores
* Listar herramientas (tools/list)
* Llamar herramientas (tools/call)
* Cambiar de servidor
* Interactuar con Claude API para generación de respuestas inteligentes.

**4.2 Servidor MCP Local**

Servidor JSON-RPC que gestiona una agenda local desde el archivo tasks\_db.json. Desarrollado como un archivo Python (mcp\_schedulizer.py), responde a métodos como list\_tasks, add\_task, generate\_schedule, etc.

**4.3 Servidor MCP Filesystem**

Implementado mediante el comando:

npx -y @modelcontextprotocol/server-filesystem workspace

Funciona por entrada/salida estándar (STDIO) para responder a comandos que interactúan con archivos del sistema.

**4.4 Servidor MCP Remoto (Google Cloud Run)**

Desplegado usando Google Cloud SDK, accesible por HTTP y habilitado para recibir JSON-RPC. Proporciona herramientas como:

* suggest\_breaks – sugerencia de descansos en agendas
* daily\_quote – frase motivacional diaria

**5. Pruebas Realizadas**

**5.1 Conexión con Servidor Local**

* Se lanzó el servidor con:

python src/mcp\_schedulizer.py

* El cliente cliente\_chatbot.py realizó llamadas con éxito a herramientas como list\_tasks.

**5.2 Conexión con Servidor Remoto (Cloud Run)**

* URL del servidor: https://mcp-schedulizer-v1-abcdef.a.run.app
* Desde el cliente, se pudo acceder y obtener resultados de suggest\_breaks y daily\_quote.

**5.3 Conexión con Servidor Filesystem**

* Se corrió el comando:

npx -y @modelcontextprotocol/server-filesystem workspace

* El cliente se conectó por stdio y pudo ejecutar herramientas como list\_files, read\_file, write\_file.

**6. Análisis de Tráfico con Wireshark**

**6.1 Captura 1 – MCP Local**

* Protocolo: HTTP en localhost:3000
* Intercambio JSON-RPC visible en texto plano
* Confirmación de envío y recepción de llamadas como list\_tasks, generate\_schedule

**6.2 Captura 2 – MCP Remoto (Cloud Run)**

* IPs de destino: 34.143.XX.2 (resueltas por nslookup)
* Se observaron paquetes HTTP POST a la URL del servidor
* JSON con parámetros como method, params, id

**6.3 Captura 3 – Filesystem (STDIO)**

* No se observa en Wireshark (no pasa por red), ya que se comunica vía stdin/stdout
* Confirmación de ejecución por consola y respuestas en terminal del cliente

**7. Validación de Resultados**

| **Prueba** | **Resultado Esperado** | **Resultado Obtenido** | **Estado** |
| --- | --- | --- | --- |
| list\_tasks en local | Lista de tareas | Correcto | ✅ |
| suggest\_breaks en remoto | JSON con descansos sugeridos | Correcto | ✅ |
| list\_files en filesystem | Archivos de la carpeta workspace | Correcto | ✅ |

Imagen wireshark uso general

Una captura de pantalla de una computadora

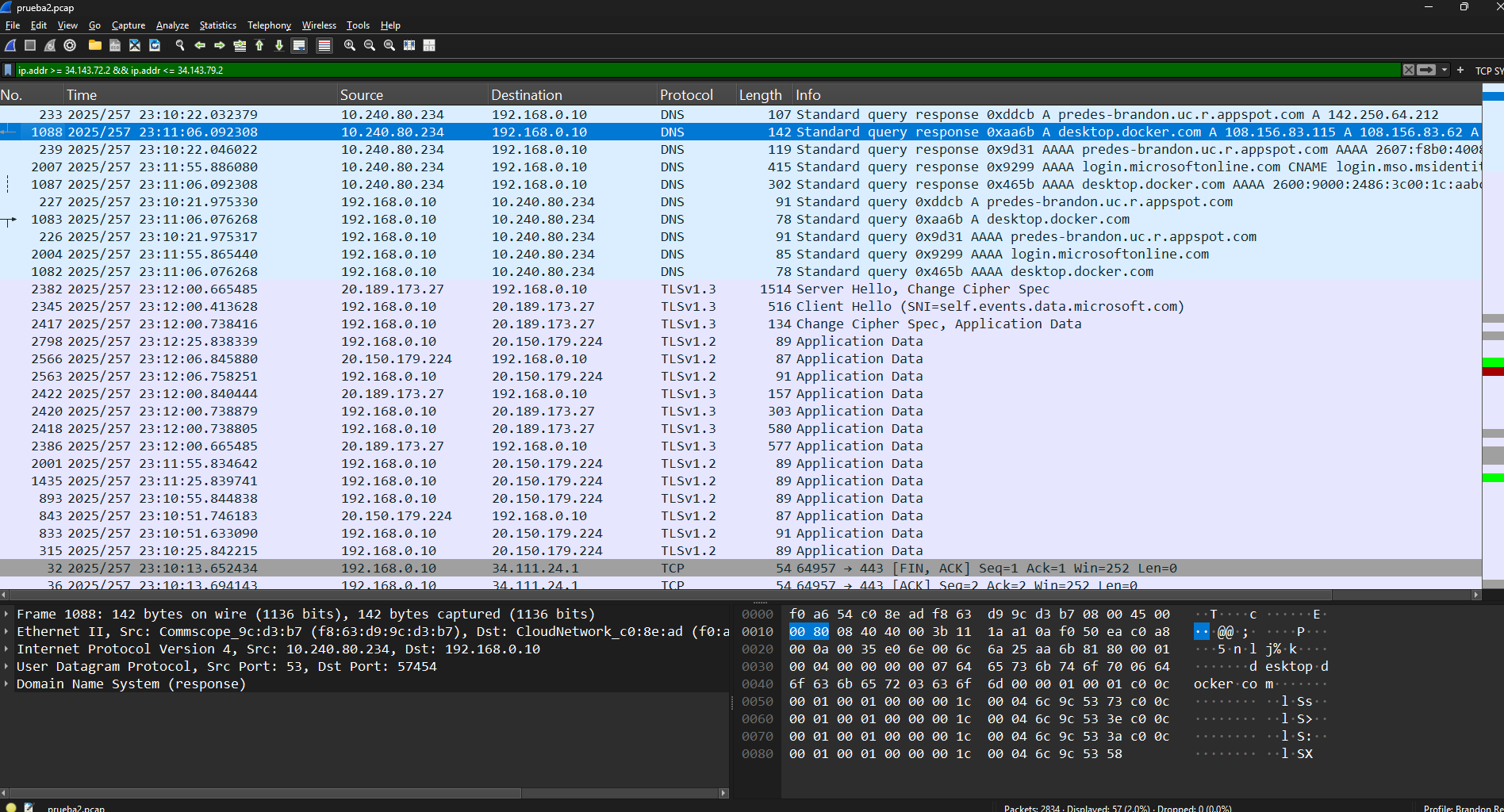
El contenido generado por IA puede ser incorrecto.

Imagen wireshark servidor local

Interfaz de usuario gráfica, Texto

El contenido generado por IA puede ser incorrecto.

Imagen wireshark servidor remoto



**8. Conclusiones**

* Se logró integrar correctamente tres tipos de servidores MCP con diferentes protocolos (HTTP, STDIO).
* El análisis de Wireshark permitió verificar que el cliente cliente\_chatbot.py está comunicándose correctamente con los servidores local y remoto.
* La modularidad del protocolo JSON-RPC facilitó el uso uniforme de herramientas, sin importar si eran remotas, locales o de archivos.
* El despliegue en Google Cloud Run probó ser efectivo para exponer servicios de manera segura y escalable.

**9. Recomendaciones**

* Para futuras versiones, se recomienda implementar autenticación por API Key para proteger el servidor remoto.
* Añadir más herramientas educativas al servidor, como un generador de horarios de estudio según materias.
* Incluir logging persistente para registrar interacciones del cliente con cada servidor.

**10. Anexos**

* Capturas de pantalla: configuración de proyecto en Google Cloud, ejecución de nslookup, interfaz del cliente, respuestas Claude
* Archivos: cliente\_chatbot.py, mcp\_schedulizer.py, tasks\_db.json, capturas .pcap, capturas Wireshark en imagen