Nombre: Carlos Alberto Valladares Guerra

Proyecto 1: Uso de un Protocolo Existente (MCP)

1. Especificación de los servidores MCP desarrollados

Durante el proyecto se implementaron e integraron cuatro servidores MCP distintos:

• Servidor Local (HTTP):
- Desarrollado en Python con Flask + SQLAlchemy.
- Funcionalidad: gestión de tareas (crear, listar, completar, posponer).
- Endpoints principales: /initialize, /describe, /run.
- Parámetros: puerto 6000, namespace 'local__'.

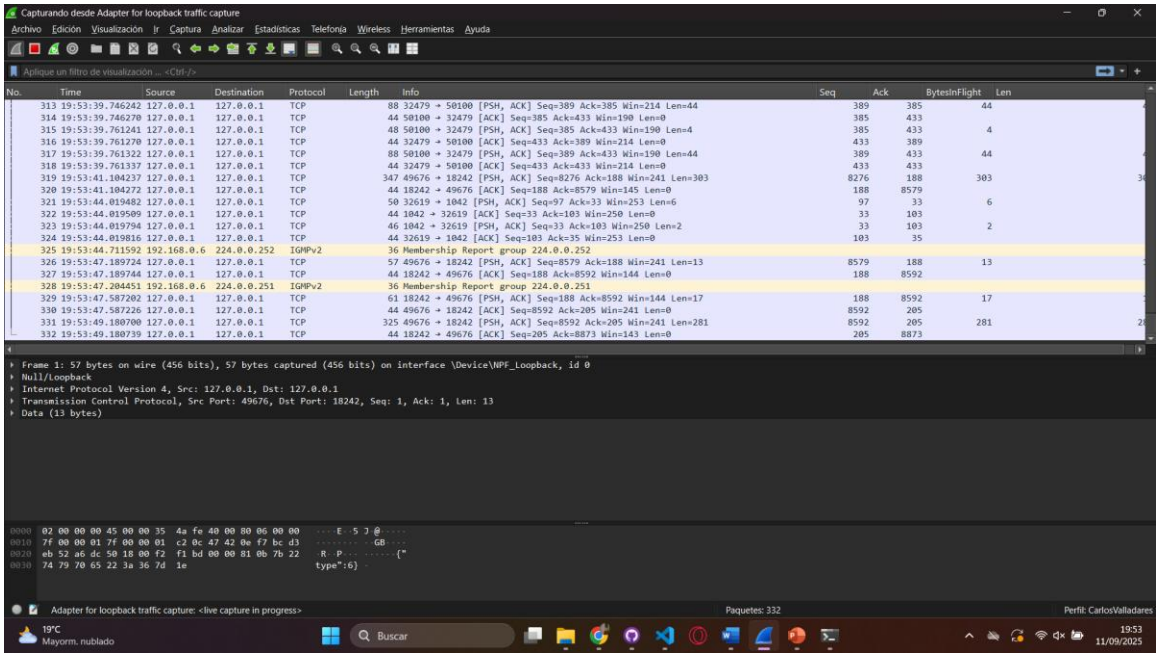
• Servidor Remoto (HTTP en Google Cloud Run):
- Implementado como microservicio en la nube.
- Funcionalidad: operaciones financieras (consultar saldos, registrar pagos).
- Endpoints: /initialize, /describe, /run.
- Parámetros: URL pública asignada por Cloud Run, namespace 'remoto__'.

• Filesystem MCP (Oficial):
- Ejecutado vía: npx -y @modelcontextprotocol/server-filesystem <directorio_root>.
- Funcionalidad: lectura, escritura y listado de archivos dentro del directorio workspace.
- Se comunicó vía STDIO, con herramientas como read_file, write_file, list_directory.
- Namespace: 'fs__'.

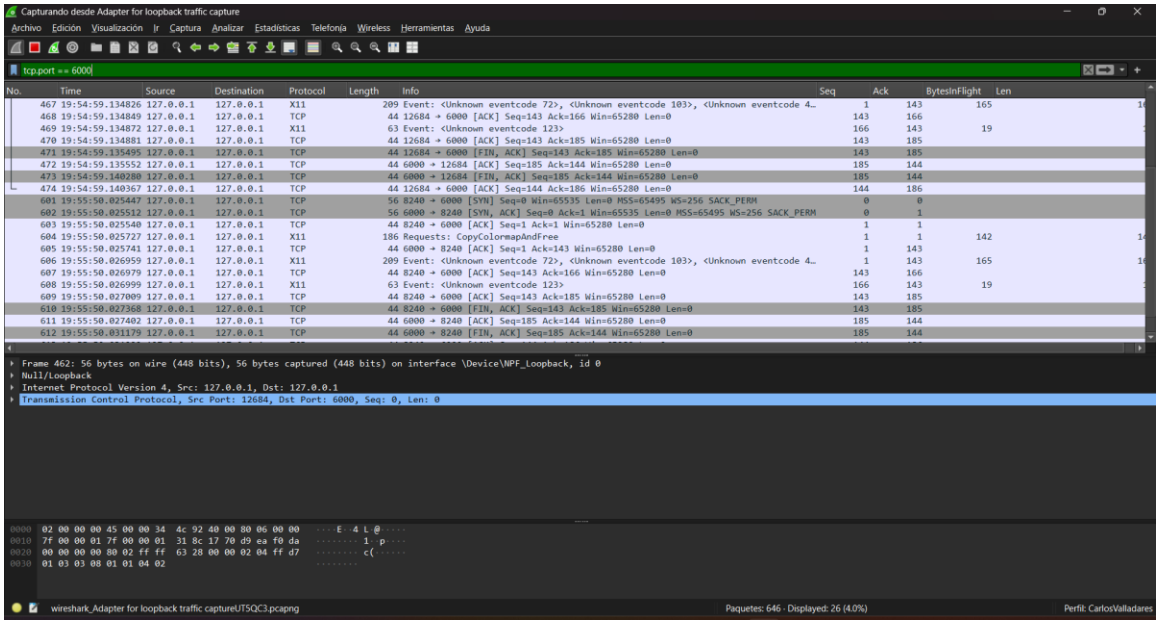
• Git MCP (Oficial):
- Ejecutado con: python -m mcp_server_git --repository ./repo_git.
- Funcionalidad: control de versiones (git status, git add, git commit, git log).
- Comunicación STDIO, exponiendo herramientas equivalentes a comandos de Git.
- Namespace: 'git__'.

2. Análisis de comunicación en red

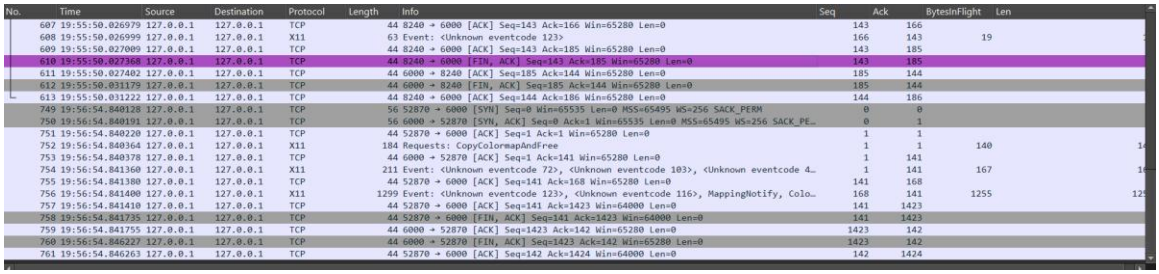
Se utilizó Wireshark y los logs de ejecución para analizar la comunicación del protocolo MCP:
Local:
• Capa de Enlace (OSI 2): transmisión de tramas Ethernet/Wi-Fi entre cliente y servidor en la red local.



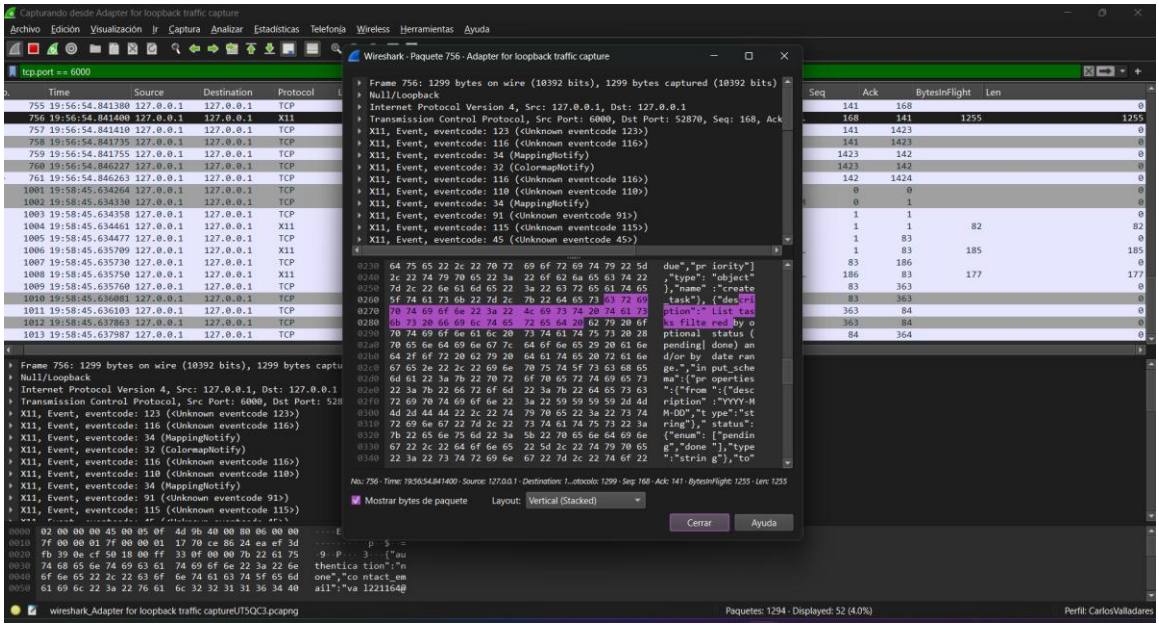
• Capa de Red (OSI 3): direccionamiento IP; en local 127.0.0.1 para servidores locales, y direcciones externas para el servidor remoto en Google Cloud Run.



• Capa de Transporte (OSI 4): uso de TCP como protocolo confiable para la entrega de mensajes.

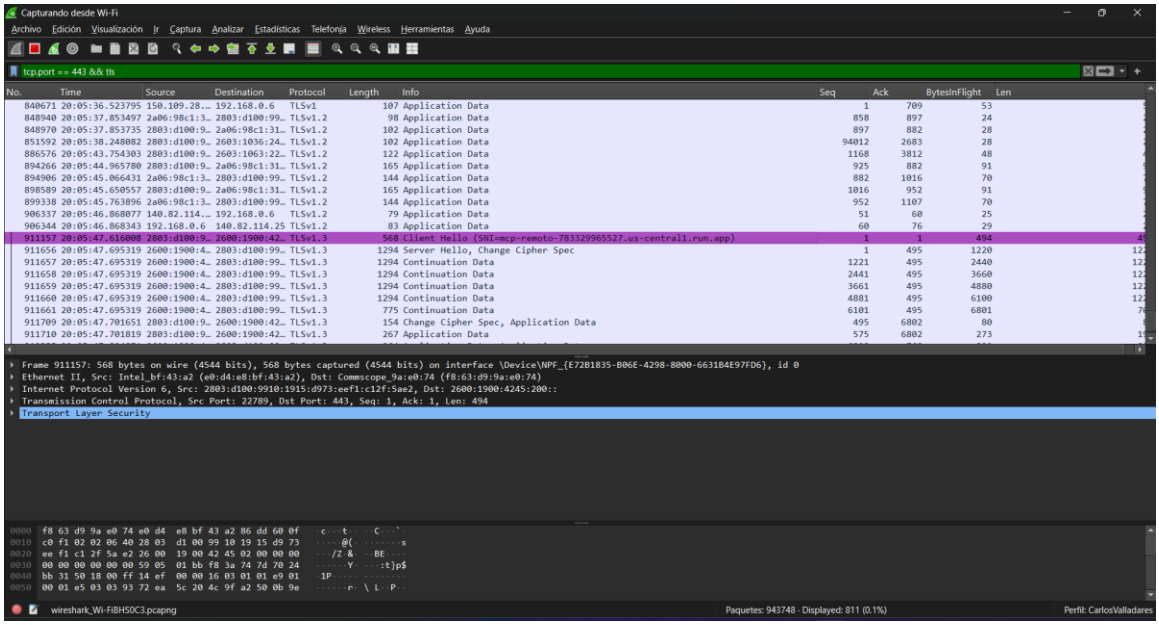


• Capa de Aplicación (OSI 7): intercambio de mensajes JSON-RPC con estructura definida:
  - initialize: negociación inicial, devuelve protocolVersion, capabilities y serverInfo.
  - describe/list_tools: descubrimiento de herramientas expuestas.
  - call_tool/run: ejecución de la herramienta especificada, con argumentos JSON.

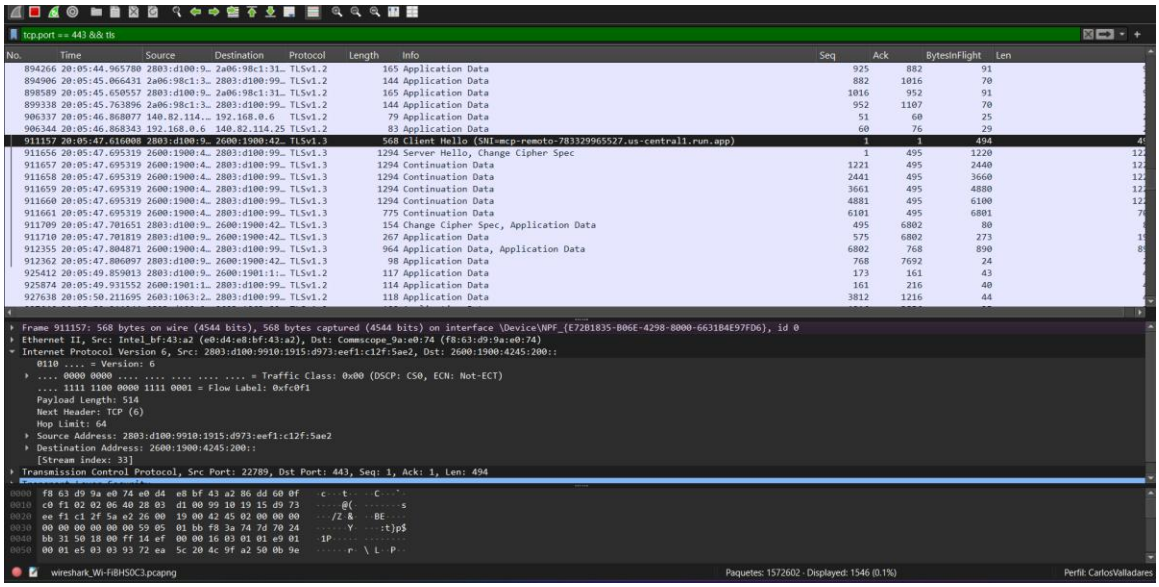


Remoto:

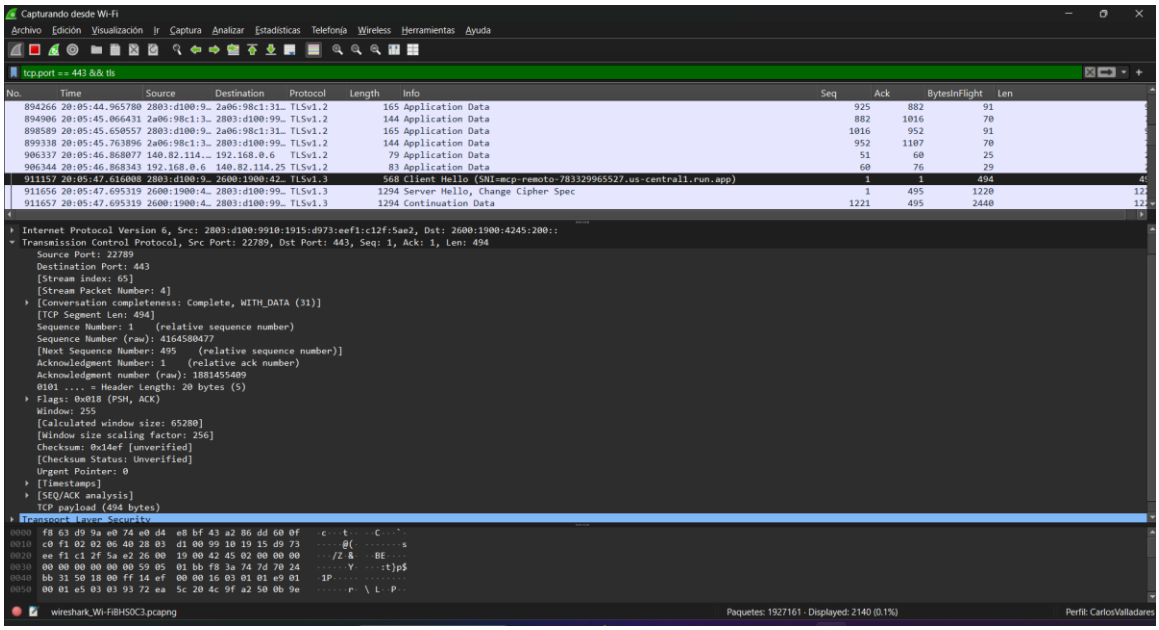
• Capa de Enlace (OSI 2): transmisión de tramas Ethernet/Wi-Fi entre cliente y servidor en la red.



• Capa de Red (OSI 3): direccionamiento IP; en local 127.0.0.1 para servidores locales, y direcciones externas para el servidor remoto en Google Cloud Run.

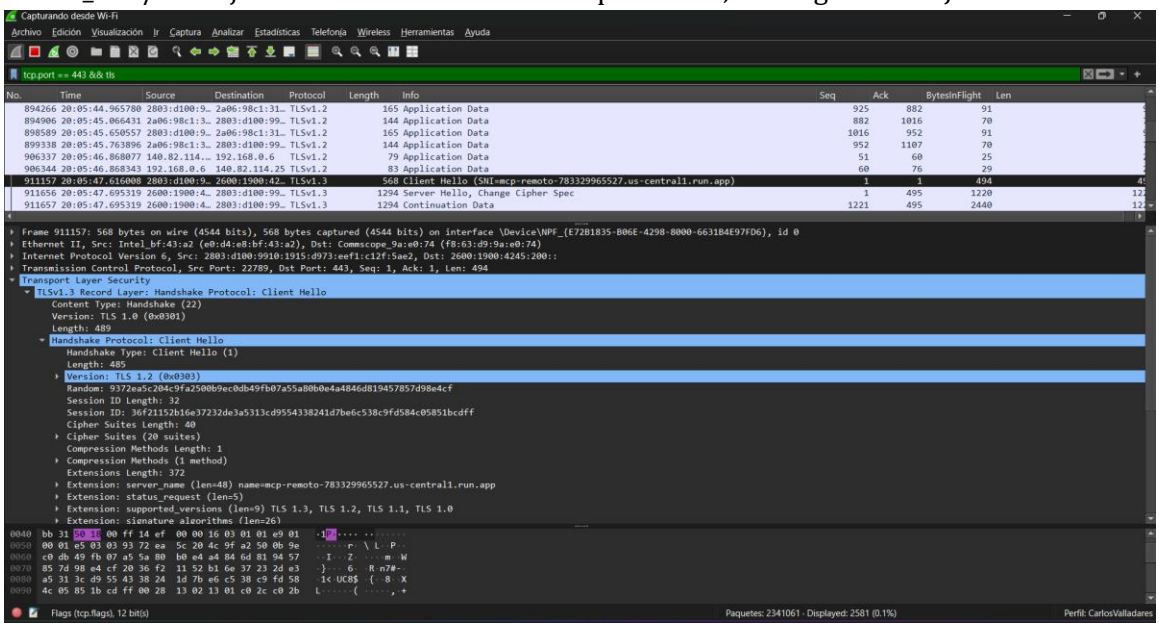


• Capa de Transporte (OSI 4): uso de TCP como protocolo confiable para la entrega de mensajes.



• Capa de Aplicación (OSI 7): intercambio de mensajes JSON-RPC con estructura definida:
- initialize: negociación inicial, devuelve protocolVersion, capabilities y serverInfo.
- describe/list_tools: descubrimiento de herramientas expuestas.

- call_tool/run: ejecución de la herramienta especificada, con argumentos JSON.



3. Conclusiones y comentarios

El proyecto permitió comprobar la utilidad del protocolo MCP como mecanismo de integración entre un chatbot y múltiples servidores especializados. Se logró unificar la gestión de tareas, operaciones financieras, manejo de archivos y control de versiones en un mismo asistente conversacional.

Comentarios principales:
- MCP facilita la interoperabilidad, pero no es plug & play: requiere inicialización y manejo de sesiones.
- Los servidores oficiales (Filesystem, Git) validan que el estándar puede aplicarse a casos reales.
- Las principales dificultades fueron compatibilidad en Windows y validación de nombres de herramientas, resueltas con configuración y sanitización.
- El uso de logs permitió mantener trazabilidad de todas las operaciones.

En conclusión, el proyecto cumplió con los objetivos académicos, integrando teoría de protocolos de red con práctica en sistemas distribuidos.