UNIVERSIDAD DEL VALLE DE GUATEMALA
Redes
Sección 30

Proyecto 1 - Uso de un protocolo existente

Paula Rebeca Barillas Alvarez, 22764

Guatemala. Septiembre 2025

Introducción

Los Large Language Models (LLMs) son un tipo de modelo de lenguaje de aprendizaje profundo que se preentrenan con grandes cantidades de datos. Para que se puedan utilizar al crear modelos de inteligencia artificial y solo pueden responder basándose en su conocimiento previo. Ya que El transformador subyacente es un conjunto de redes neuronales que consta de un codificador y un decodificador con capacidades de autoatención. El codificador y el decodificador extraen significados de una secuencia de texto y comprenden las relaciones entre las palabras y las frases que contiene. Asimismo, los chatbots y agentes proporcionan herramientas a los LLMs para aumentar su capacidad de interacción con el mundo real.

En este proyecto se utiliza el Model Context Protocol (MCP) para proveer tools a los agentes y chatbots, independientemente del LLM utilizado. Se implementaron 6 servidores MCP independientes en un solo chatbot, combinando soluciones locales y remotas.

Arquitectura del Sistema

Componentes Principales:

- **Anfitrión (Host)**: Chatbot en consola que coordina todos los MCPs
- **Cliente MCP**: Interfaz que mantiene conexiones con servidores
- **Servidores MCP**: Herramientas que ejecutan acciones específicas

Tecnologías Utilizadas:

- **Lenguaje**: Node.js/JavaScript
- **LLM**: Antropic - Claude
- **Protocolo**: JSON-RPC 2.0 sobre HTTP/HTTPS
- **Despliegue Remoto**: Cloudflare Workers
- **Análisis de Red**: Wireshark

Servidores MCP Implementados

MCPs Locales (STDIO)

1. FilesystemMCP

- **Descripción**: Herramienta que permite interactuar de manera segura con sistemas de archivos locales
- **Basado en**: @modelcontextprotocol/server-filesystem
- **Parámetros**:
    - path: Ruta donde se desea operar
    - content: Contenido de texto para escritura
- **Endpoints**:
    - filesystem:create_directory
    - filesystem:write_file
    - filesystem:read_text_file
    - filesystem:list_directory

2. GitMCP

- **Descripción**: Servidor para integración con herramientas de Git, proporcionando asistencia en desarrollo
- **Basado en**: @cyanheads/git-mcp-server
- **Parámetros**:
    - path: Directorio del repositorio
    - message: Mensaje del commit
    - remote_url: URL del repositorio remoto
    - branch: Rama de trabajo
- **Endpoints**:
    - git_set_working_dir
    - git_init
    - git_add
    - git_commit

    - git_log
    - git_status

3. KitchenMCP

- **Descripción**: Servidor personalizado para gestión de recetas y cocina
- **Implementación**: Desarrollado localmente
- **Funcionalidades**:
    - Búsqueda de recetas
    - Sustitución de ingredientes
    - Cálculos nutricionales
- **Endpoints:**
    - recommend_by_mood_and_season: Recomienda comidas basadas en estado de ánimo y temporada
    - suggest_utensils_for_recipe: Sugiere utensilios necesarios para una receta específica
    - get_foods: Obtiene lista completa de alimentos disponibles
    - suggest_recipe_by_diet: Sugiere recetas por tipo de dieta (vegana, keto, mediterránea, etc.)
    - suggest_ingredient_substitution: Encuentra sustitutos para ingredientes específicos
    - get_food_by_name: Busca información de un alimento específico
    - search_foods: Busca alimentos por criterios nutricionales
    - get_ingredients: Lista de ingredientes disponibles
    - get_recipe_suggestions: Sugerencias de recetas basadas en contenido nutricional
    - get_recipes: Obtiene todas las recetas disponibles
    - get_recipes_by_ingredients: Encuentra recetas por ingredientes específicos

4. TransfermarktMCP (Compañero: Derek Arreaga)

- **Descripción**: Servidor para consultas de datos de fútbol (Mercado / fichaje de futbolistas)
- **Implementación**: TypeScript
- **Funcionalidades**:
    - Información de jugadores
    - Estadísticas de equipos
    - Datos de transferencias
- **Endpoints:**
    - get-player-transfers: Obtiene todas las transferencias de un jugador específico
    - get-player-stats: Obtiene estadísticas de un jugador en un club específico
    - get-career-path: Obtiene la carrera completa de un jugador (clubes y años)
    - get-last-transfer: Obtiene la transferencia más reciente de un jugador
    - players-in-clubs: Encuentra jugadores que han jugado en TODOS los clubes especificados
    - player-career-summary: Obtiene resumen completo de la carrera de un jugador

5. SleepCoachMCP (Compañera: Fabiola Contreras)

- **Descripción**: Servidor para análisis y consejos de patrones de sueño
- **Implementación**: Python
- **Funcionalidades**:
    - Análisis de calidad del sueño
    - Recomendaciones personalizadas
    - Seguimiento de patrones
- **Endpoints:**
    - create_user_profile: Crea un perfil personalizado con hábitos de sueño actuales del usuario
    - analyze_sleep_pattern: Analiza el patrón de sueño actual y detecta problemas específicos

- get_personalized_recommendations: Genera recomendaciones personalizadas basadas en el perfil
- create_weekly_schedule: Crea un horario semanal optimizado con rutinas personalizadas
- quick_sleep_advice: Proporciona consejos rápidos basados en consultas específicas

MCP Remoto (HTTP/HTTPS)

6. RemoteMCP

- **Descripción**: Servidor desplegado en Cloudflare Workers para funcionalidades remotas
- **URL**: https://mcp-remote.paulabarillas.workers.dev
- **Implementación**: Cloudflare Workers
- **Funcionalidades**:
  - Servicios de tiempo en tiempo real
  - Generación de contenido aleatorio
  - Datos de entretenimiento (Taylor Swift)

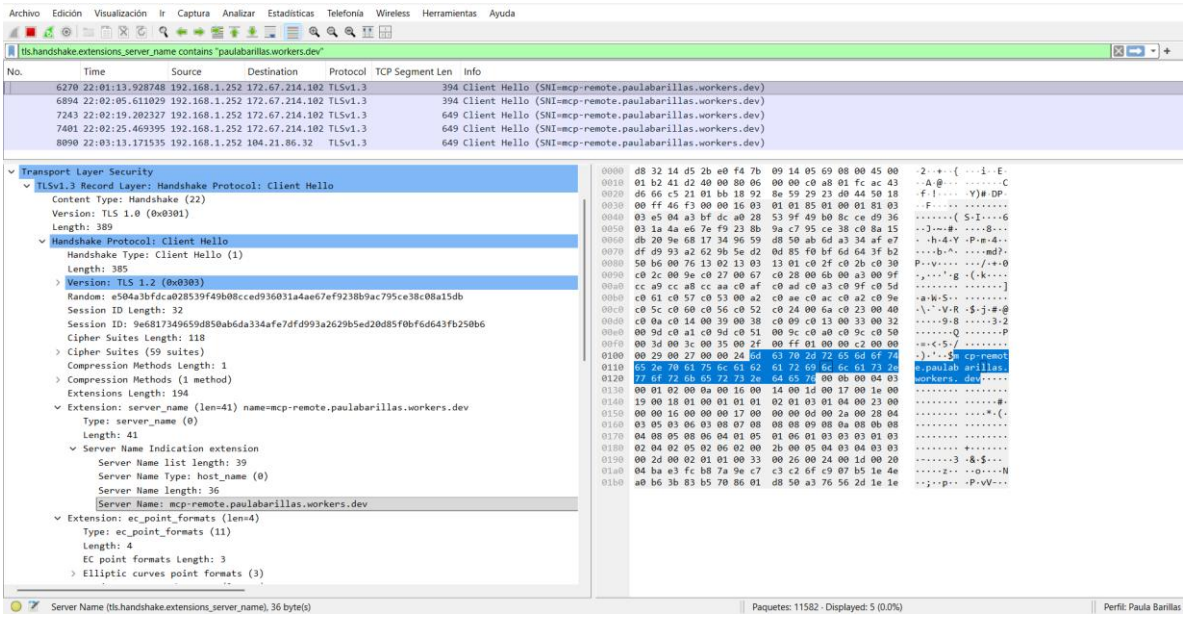
Endpoints Disponibles:

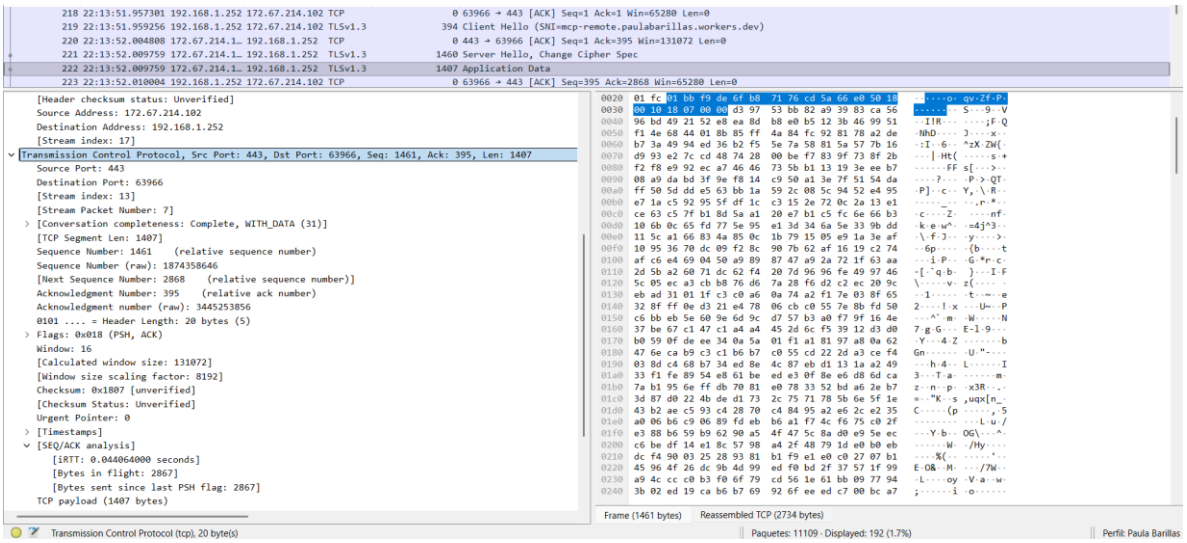
- get_time: Obtiene la hora actual en UTC o zona horaria específica
- lucky_number: Genera un número de la suerte aleatorio entre 1 y 100
- taylor_lyric: Obtiene letra y título aleatorio de canciones de Taylor Swift

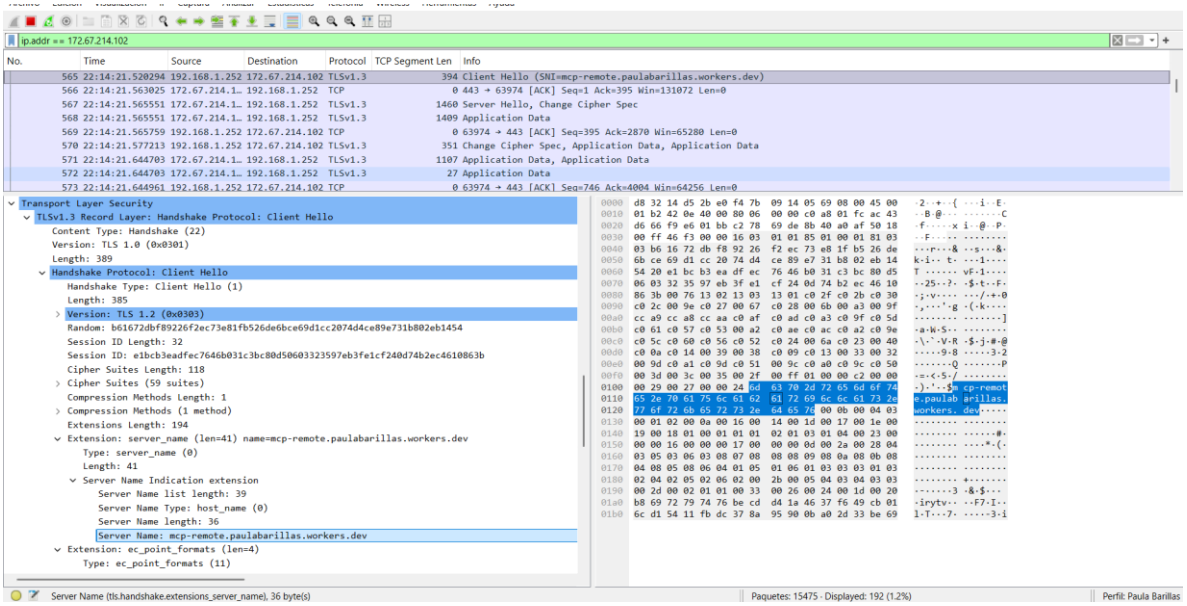
Análisis de Comunicación con Wireshark

Configuración de Captura

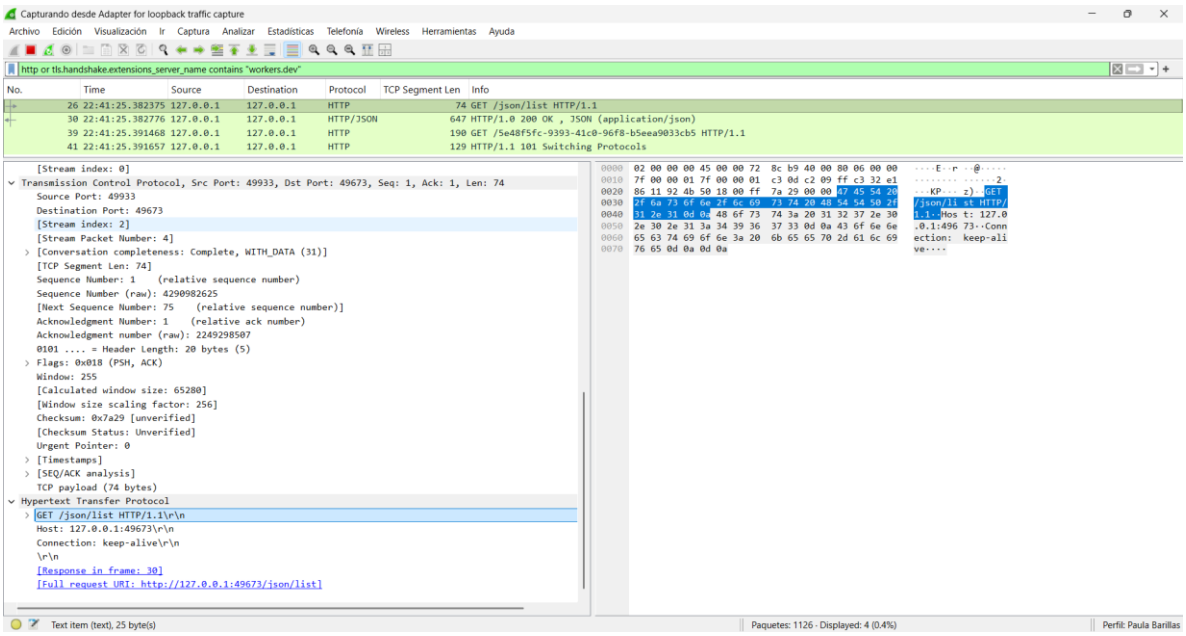
- **Filtros utilizados:**
    - ip.addr == 172.67.214.102
    - tls.handshake.extensions_server_name contains "workers.dev"
    - http or tcp.port == 49673
- **Interfaces:**
    - Adaptador WiFi (tráfico remoto)
    - Loopback (tráfico local)







- **Source/Destination:** IPs del cliente MCP y Cloudflare Workers
- **Protocol:** TLSv1.3 | cifrado moderno



Para MCPs (HTTP - Visible)

- Capa de Enlace: Ethernet II
- Capa de Red: IPv4 (127.0.0.1 ↔ 127.0.0.1)
- Capa de Transporte: TCP (puerto 49673)
- Capa de Sesión: HTTP/1.1 (sin cifrado)
- Capa de Aplicación: JSON-RPC VISIBLE

Para MCPs Remotos (HTTPS - Cifrado)

- Capa de Enlace: Ethernet II
- Capa de Red: IPv4 (192.168.1.252 ↔ 172.67.214.102)
- Capa de Transporte: TCP (puerto 443)
- Capa de Sesión: TLS 1.3 (cifrado)
- Capa de Aplicación: HTTP/JSON-RPC CIFRADO

En la captura de tráfico con Wireshark se observa una jerarquía diferencial de visibilidad según el modelo OSI, como se puede ver en la comunicación MCP remota cifrada. En el caso de mi servidor remoto (Cloudflare Workers), la capa de aplicación JSON-RPC/HTTP permanece completamente invisible debido al cifrado TLS 1.3, que actúa como un túnel criptográfico el cuál protege todo el contenido de los mensajes MCP.

Sin embargo, TLS en la capa de presentación/sesión si se observa visible en Wireshark, permitiendo observar el handshake completo, intercambio de certificados, negociación de cipher suites, y los metadatos de las sesiones cifradas. Esto se debe a que TLS necesita operar de manera transparente para establecer la comunicación segura, exponiendo su protocolo de establecimiento mientras mantiene el payload cifrado. Asimismo, las capas inferiores (TCP/IP y Ethernet) son totalmente transparentes en Wireshark, ya que muestra la información completa sobre enrutamiento IP, puertos TCP, control de flujo, acknowledgments, y tramas Ethernet.

Respuestas JSON-RPC

1. Servidor MCP Remoto (HTTP) - Taylor Swift Lyrics (simplificado)

REQUEST:


```
{
  "method": "taylor_lyric",
  "params": {}
}
```


RESPONSE:


```
{
  "title": "exile (Ft. Bon Iver)",
  "lyric": "You never turned things around"
}
```


2. Servidor MCP Local (STDIO) - Sustitución de Ingredientes

REQUEST:


```
{
  "jsonrpc": "2.0",
  "id": 2,
  "method": "tools/call",
  "params": {
    "name": "suggest_ingredient_substitution",
    "arguments": {
      "ingredient": "rice"
```

```
    }
  }
}
```


RESPONSE:


```
{
  "result": {
    "content": [
      {
        "type": "text",
        "text": "{\n  \"ingredient\": \"rice, brown, long-grain\",\n  \"substitutions\": [\n  \"Barley, pearled, cooked\",\n  \"Oil, rice bran\",\n  \"Quinoa, uncooked\",\n  \"Wild rice, raw\"\n  ]\n}"
      }
    ]
  },
  "jsonrpc": "2.0",
  "id": 2
}
```


3. Servidor MCP Local (STDIO) - Utensilios de Cocina

REQUEST:


```
{
  "jsonrpc": "2.0",
  "id": 3,
  "method": "tools/call",
```

```
 "params": {
  "name": "suggest_utensils_for_recipe",
  "arguments": {
   "recipe_name": "lasaña"
  }
 }
}
```


RESPONSE:


```
{
 "result": {
  "content": [
   {
    "type": "text",
    "text": "{\n  \"recipe\": \"lasaña\",\n  \"utensils\": [\n   \"knife\",\n   \"cutting board\",\n  \"spoon\",\n   \"fork\",\n   \"bowl\",\n   \"pot\",\n   \"pan\",\n   \"colander\",\n   \"measuring cups\",\n   \"measuring spoons\",\n   \"mixing bowls\",\n   \"whisk\",\n   \"spatula\",\n  \"tongs\",\n   \"oven mitts\",\n   \"peeler\",\n   \"grater\",\n   \"ladle\",\n   \"can opener\",\n  \"tray\",\n   \"storage container\"\n  ]\n}"
   }
  ]
 },
 "jsonrpc": "2.0",
 "id": 3
}
```

Conclusiones

Con este proyecto logre entender que el protocolo MCP representa una forma diferente de ver la arquitectura de los sistemas de IA esto debido a que no se limita a integrar herramientas de manera rígida, sino que las desacopla y permite que funcionen de alguna forma "más sencilla". Al trabajar con distintos tipos de servidores, tanto locales como el remoto pude darme cuenta de las diferencias en la comunicación en donde en una se entiende la transparencia que facilita el análisis en entornos de desarrollo, y por otro la seguridad que se logra en escenarios de producción con el uso de TLS (como se observo con wireshark). También el hacer lo posible de no hacer uso de SDKs me ayudó a profundizar en cómo realmente funciona JSON-RPC y a entender que MCP está lo suficientemente bien definido o elaborado como para ser implementado sin depender de librerías externas o algo que lo complique. Esto me dejó claro que el protocolo no solo es útil en la práctica o bien solo ahorita para el proyecto sino que también tiene potencial para crecer como un estándar en arquitecturas más robustas. En conclusión, este proyecto me permitió comprobar de manera directa la importancia o bien como funciona un MCP y como abre el camino hacia sistemas de IA más modulares, seguros e interoperables, algo que considero esencial para el futuro.

Comentarios del proyecto

Honestamente, al principio creo que llegó a ser abrumador (personalmente) debido a que las instrucciones no fueron de todo claras y creo que falto mejor explicación (si me confundí varias veces) y por lo mismo faltaron aclaraciones; ya que también nunca nos dijeron si estaban bien los MCP seleccionado. Como desafíos, habían funciones o bien el caso del mcp de Git que no me funcionaba o bien no existían, y tuve que ir probando varias alternativas por lo cual acá creo que fue donde entendí mejor el buscar alterantivas y pensar bien una lógica para hacer uso de varios MCP. Ahora bien, yo hice uso de Anthropic; y personalmente quedo satisfecha con el uso (aunque si me cobraron) creo que fue bastante intuitivo. En general considero que es un buen proyecto y tiene bastante potencial, aunque personalmente trataría de explicarlo mejor, aunque ya esto es personal me hubiera gustado que se viera mejor visualmente y hacerle mejoras a mi chatbot.

Referencias:

JSON-RPC 2.0 specification. (s/f). Jsonrpc.org. https://www.jsonrpc.org/specification

¿Qué son los modelos de lenguaje de gran tamaño? (s/f). Amazon.com. https://aws.amazon.com/es/what-is/large-language-model/

Wireshark • display filter reference: Miscabling protocol. (s/f). Wireshark. https://www.wireshark.org/docs/dfref/m/mcp.html

Anexo:

Repositorio MCP Local : https://github.com/paulabaal12/kitchen-mcp

Repositorio Chatbot + MCP remoto: https://github.com/paulabaal12/chatbot-mcp