DELVALLE
GRUPO EDUCATIVO

Proyecto 1
Redes

Hugo Rivas - 22500

9.1 Protocolo y transporte

- Protocolo lógico: MCP (Model Context Protocol) sobre JSON-RPC 2.0.
- Transporte: HTTP(S).
    - Servidor remoto (Cloud Run): HTTPS y opción de respuesta SSE (Server-Sent Events) para streaming.
    - Servidor local DNS: expuesto internamente al host (embebido vía FastMCP); mismo contrato MCP de tools.
- Negociación de versión: el cliente envía initialize con protocolVersion: "2025-06-18".
- Métodos JSON-RPC implementados: initialize, tools/list, tools/call.

9.2 Endpoints

A) Servidor remoto server_remote.py

Endpoint	Método	Descripción
/	GET	Health. Responde JSON con ok, kind, mount, tools.
/mcp	POST	Punto MCP. Recibe initialize, tools/list, tools/call (JSON-RPC 2.0).
/mcp	OPTIONS	Preflight CORS.
/mcp	POST + Accept: text/event-stream	Respuesta SSE (líneas data: {...} y cierre data: [DONE]).

Cabeceras relevantes

- Content-Type: application/json
- Accept: application/json o text/event-stream
- Mcp-Session-Id (opcional; expuesto vía CORS)
- CORS de respuesta:
  Access-Control-Allow-Origin: *,
  Access-Control-Allow-Headers: *,
  Access-Control-Allow-Methods: GET,POST,OPTIONS,
  Access-Control-Expose-Headers: Mcp-Session-Id.

Variables de entorno: PORT.

B) Servidor local DNS servidor.py

- Servidor MCP con FastMCP("MCP-DNS"). En este proyecto se consume embebido (sin HTTP), pero sus tools siguen el contrato MCP.

- Log NDJSON: DNS_MCP_LOG (por defecto dns_mcp.log.jsonl).

9.3 Mensajes JSON-RPC (esquemas y ejemplos)

initialize

Request


```
{
  "jsonrpc": "2.0",
  "id": "uuid-1",
  "method": "initialize",
  "params": { "protocolVersion": "2025-06-18", "capabilities": {} }
}
```


Response


```
{
  "jsonrpc": "2.0",
  "id": "uuid-1",
  "result": { "capabilities": {} }
}
```


tools/list

Request


```
{ "jsonrpc": "2.0", "id": "uuid-2", "method": "tools/list", "params": {} }
```


Response (remoto)


```
{
  "jsonrpc": "2.0",
  "id": "uuid-2",
  "result": {
    "tools": [
      { "name": "echo", "description": "", "inputSchema": { "type": "object", "properties": { "texto": { "type": "string" }}, "required": ["texto"] } },
      { "name": "morse", "description": "", "inputSchema": { "type": "object", "properties": { "texto": { "type": "string" }}, "required": ["texto"] } },
```

{ "name": "demorse", "description": "", "inputSchema": { "type": "object", "properties": { "codigo": { "type": "string" }}, "required": ["codigo"] } }
]
}
}

tools/call

Request (ej. remoto: morse)

{
"jsonrpc": "2.0",
"id": "uuid-3",
"method": "tools/call",
"params": { "name": "morse", "arguments": { "texto": "SOS prueba" } }
}

Response (formato FastMCP)

{
"jsonrpc": "2.0",
"id": "uuid-3",
"result": {
"content": [{ "type": "text", "text": "... --- ... / .--. .-. ..- . -... .-" }]
}
}

Respuesta SSE (cuando Accept: text/event-stream)

data: {"jsonrpc":"2.0","id":"uuid-3","result":{...}}
data: [DONE]

Errores manejados por el cliente

- HTTP 429/503 → reintento tras ~1.5 s.
- JSON-RPC con error (code, message).
- Timeouts HTTP (15 s).

9.4 Catálogo de herramientas (tools)

A) Remoto server_remote.py

1. echo(texto: str) -> str
   Retorna echo: <texto>.

2. morse(texto: str) -> str
   Codifica a Morse (mayúsculas y sin acentos). Letras separadas por espacio; palabras por /.

3. demorse(codigo: str) -> str
   Decodifica Morse; acepta | como separador alterno de palabra.

B) Local DNS servidor.py

Todas registran evento en DNS_MCP_LOG con tool, dominio, dur_ms y tamaño de salida.

1. ping() -> str → "pong".

2. salud_dns(dominio: str) -> Dict
   Consultas recursivas (A/AAAA/NS/SOA), consultas directas a autoritativos (A/AAAA/NS/SOA), detección de wildcard, heurística de TTLs desbalanceados, CNAME colgante.
   Salida principal:
   dominio, recursivo{A,AAAA,NS,SOA}, autoritativo{A,AAAA,NS,SOA}, hallazgos[{tipo,severidad,detalle}].

3. correo_politicas(dominio: str) -> Dict
   Lee MX, busca SPF (TXT apex) y DMARC (_dmarc.).
   Salida: mx, spf, dmarc, hallazgos.

4. estado_dnssec(dominio: str) -> Dict
   DS en padre, DNSKEY (algoritmos), validación práctica de RRSIG(SOA) contra DNSKEY con consultas autoritativas (UDP y fallback TCP si TC).
   Salida: tiene_ds_en_padre, dnskey_algoritmos, soa_firmada_valida (true/false/null), detalles, hallazgos.

5. propagacion(dominio: str, resolutores?: List[str]) -> Dict
   Compara A/AAAA/NS (y muestra de TXT) entre resolutores (por defecto: 1.1.1.1, 8.8.8.8, 9.9.9.9).
   Salida: resolutores, respuestas{por IP}, diferencias{A,AAAA,NS}.

9.5 Requisitos del cliente (host.py)

- Headers: Content-Type: application/json, Accept: application/json, text/event-stream, Mcp-Session-Id (si existe).

- Timeout HTTP: 15 s.

- Reintentos: para 429/503.
- Parsing SSE: se consumen líneas data: y se usa el último JSON válido.
- Log local de chat: chat_log.jsonl.

9.6 Seguridad

- Capa de transporte: TLS 1.3 (Cloud Run).
- Autenticación de aplicación: no implementada (solo cifrado TLS).
- CORS del remoto: abierto a * y expone Mcp-Session-Id.

8) Análisis de la comunicación cliente/servidor (Wireshark)

Host local: 192.168.0.14
Servidor remoto (Cloud Run): 34.143.73.2
Puerto: 443/TCP (TLS 1.3)
Filtros usados:

- tls.handshake.extensions_server_name == "mcp-hello-remote-py-145050194840.us-central1.run.app"
- http.host == "mcp-hello-remote-py-145050194840.us-central1.run.app"
- tcp.stream eq <N> para seguir cada flujo.

Secuencia observada (por flujo TCP)

1. Descubrimiento/Health (no es JSON-RPC) — tcp.stream eq 5

- Request: GET / HTTP/1.1

Response: 200 OK con JSON:

{"ok":true,"kind":"mcp-streamable-http","mount":"/mcp","tools":["echo(texto)","morse(texto)","demorse(codigo)"]}

- Propósito: *health check* y metadatos del servidor. No participa en JSON-RPC; sirve como verificación inicial.

2. Sincronización (JSON-RPC “initialize”) — tcp.stream eq 11

Request (POST /mcp):

{"jsonrpc":"2.0","id":"<uuid>","method":"initialize","params":{"protocolVersion":"2025-06-18","capabilities":{}}}

- Response (SSE text/event-stream):

Caso observado:

{"jsonrpc":"2.0","id":"<uuid>","error":{"code":-32602,"message":"Invalid request parameters","data":""}}

  -
  - El servidor MCP responde por SSE con eventos event: message y línea data: { ... } que contiene el resultado o error JSON-RPC.

- Clasificación: Mensaje de sincronización/negociación. Establece versión de protocolo y capacidades.

3. Descubrimiento de herramientas (JSON-RPC “tools/list”) — tcp.stream eq 12

Request (POST /mcp):

{"jsonrpc":"2.0","id":"<uuid>","method":"tools/list","params":{}}
Response (SSE):

{"jsonrpc":"2.0","id":"<uuid>","result":{"tools":[
{"name":"echo","inputSchema":{...},"outputSchema":{...}},
{"name":"morse",...},
{"name":"demorse",...}
]}}

- Clasificación: Solicitud/Petición (del cliente) y Respuesta (del servidor) JSON-RPC.

4. Ejecución de herramienta (JSON-RPC “tools/call” → morse) — tcp.stream eq 27

Request:

{"jsonrpc":"2.0","id":"<uuid>","method":"tools/call",
"params":{"name":"morse","arguments":{"texto":"Hola mundo otra vez"}}}

-

Response (SSE):

{"jsonrpc":"2.0","id":"<uuid>",
"result":{"content":[{"type":"text","text":".... --- .-.. .- / -- ..- -. -.. --- / --- - .-. .- / ...- . --.."}],
"structuredContent":{"result":".... --- .-.. .- / ..."},"isError":false}}

-
- Clasificación: Solicitud/Petición y Respuesta JSON-RPC con resultado.

5. Ejecución de herramienta (JSON-RPC “tools/call” → demorse) — tcp.stream eq 43

Request:

{"jsonrpc":"2.0","id":"<uuid>","method":"tools/call",
"params":{"name":"demorse","arguments":{"codigo":".... --- .-.. .- / -- ..- -. -.. --- / --- - .-. .- / ...- . --.."
}}}

-

Response (SSE):

{"jsonrpc":"2.0","id":"<uuid>",
"result":{"content":[{"type":"text","text":"HOLA MUNDO OTRA VEZ"}],"isError":false}}

- Clasificación: Solicitud/Petición y Respuesta JSON-RPC con resultado.

Tipo	Método / Estructura	Ejemplo en captura	Notas
Sincronización	method: "initialize"	tcp.stream eq 11	Negocia versión/capacidades. Respuesta puede ser result o error.
Solicitud/Petición	method: "tools/list"	tcp.stream eq 12	Descubre herramientas disponibles.
Solicitud/Petición	method: "tools/call"	tcp.stream eq 27 (morse), eq 43 (demorse)	Invoca herramienta con params.name y params.arguments.
Respuesta (éxito)	{"result": ...}	streams 12, 27, 43	Enviado por el servidor como SSE (event: message, data: {...}).
Respuesta (error)	{"error": {"code":..., "message":...}}	stream 11	Semántica JSON-RPC estándar (code -32602 = parámetros inválidos).
Fuera de JSON-RPC	GET / (health)	stream 5	Solo información de estado y metadatos.

Detalles de aplicación: el servidor usa Content-Type: application/json para el *request* y text/event-stream (SSE) para el *response*, por eso las respuestas llegan como eventos event: message con la carga real en la línea data: { ... }.

¿Qué ocurre en cada capa (en base al punto 8)?

Capa de Enlace (IEEE 802.11/Wi-Fi)

- Tramas 802.11 entre el host y el AP; el AP hace bridging hacia Ethernet/Internet.
- Se observan retransmisiones y control de ventana a nivel MAC según la calidad del enlace (no mostrado en las capturas finales, pero implícito en Wi-Fi).
- La NIC entrega frames al stack IP; a partir de aquí el análisis visible en Wireshark muestra ya IP/TCP/TLS.

Capa de Red (IP)

- Paquetes IPv4 desde 192.168.0.14 (host) hacia 34.143.73.2 (Cloud Run).
- El CPE realiza NAT (traducción a IP pública).

- No hay fragmentación visible; MTU suficiente para TLS/HTTP.
- El enrutamiento intermedio no se ve en las capturas locales, pero el TTL decrece hop a hop hasta el destino.

Capa de Transporte (TCP)

- 3-way handshake a 443/tcp: SYN → SYN/ACK → ACK.
- Control de flujo: números de secuencia/ACK, ventana deslizante, *delayed ACKs* y retransmisiones si hay pérdida.
- Encima de TCP corre TLS 1.3:
    - ClientHello con SNI mcp-hello-remote-py-145050194840.us-central1.run.app.
    - Negociación de cifrados y Change Cipher Spec; a partir de ahí, datos cifrados.
    - Para poder ver HTTP/JSON en claro en Wireshark se usó el (Pre)-Master-Secret log (SSLKEYLOGFILE), permitiendo decrifrado local.
    - El servidor anuncia Alt-Svc: h3=":443" (HTTP/3 disponible), pero la sesión observada usa HTTP/1.1 sobre TLS.

Capa de Aplicación

- HTTP/1.1:
    - GET / (health) con Content-Type: application/json.
    - POST /mcp con Content-Type: application/json (request) y respuesta como text/event-stream (SSE).
- JSON-RPC 2.0 transportado dentro del cuerpo HTTP:
    - Sincronización: initialize (negocia versión/capacidades).
    - Descubrimiento: tools/list.
    - Ejecución: tools/call con params.name y params.arguments.
    - Respuestas: éxito {"result":...} o error {"error":{code,message}}, siempre correlacionadas por el mismo id.

- En las herramientas invocadas:
    - morse(texto) devuelve la codificación en Morse.
    - demorse(codigo) devuelve el texto plano.
    - Los resultados se entregan como contenido estructurado (content/structuredContent) dentro del objeto result.

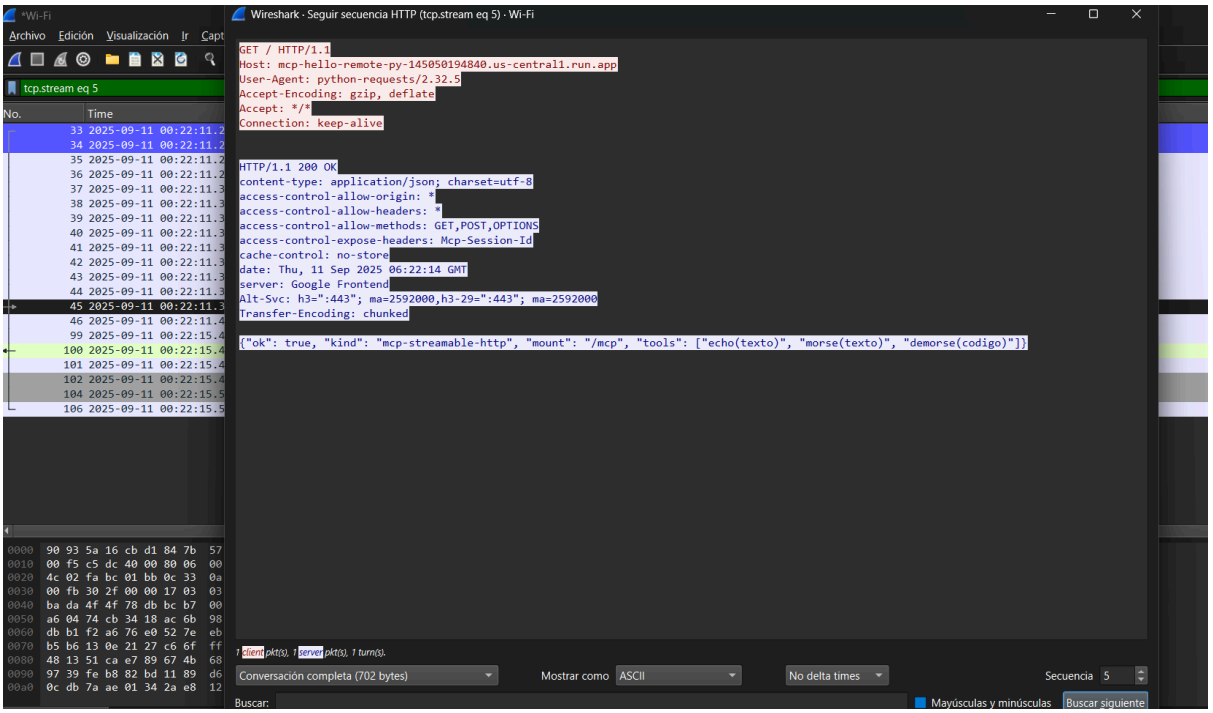
Conclusiones

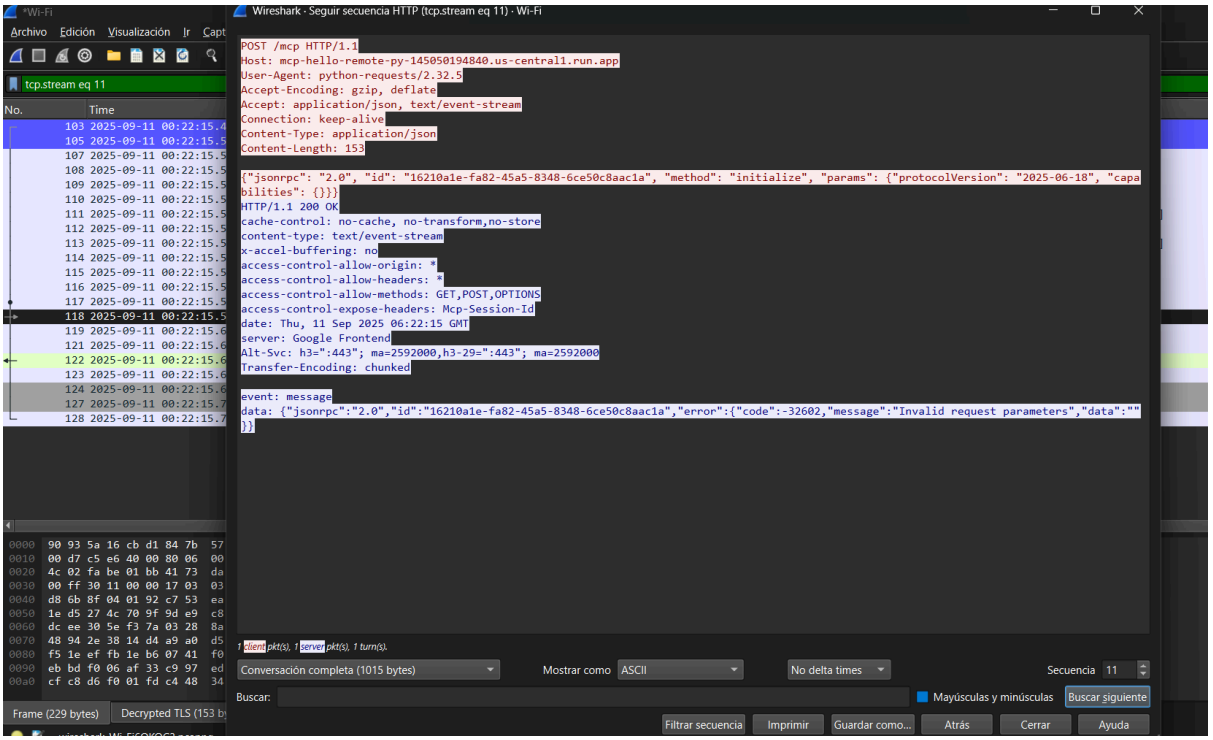
En conjunto, el proyecto logró implementar y validar un ecosistema MCP extremo a extremo: un cliente (host.py), un servidor remoto HTTP (herramientas echo/morse/demorse) y un servidor local para diagnóstico DNS, todo orquestado mediante JSON-RPC 2.0 sobre HTTPS/TLS 1.3. La instrumentación con Wireshark permitió identificar con precisión la sincronización (initialize), las peticiones (tools/list, tools/call) y sus respuestas correlacionadas por id, además de confirmar el uso de SSE para la entrega de resultados. En el plano práctico, se resolvieron fricciones de entorno (Windows/venv, rutas y SNI) y se demostró la utilidad de registros (chat_log.jsonl) y del sslkeys.log para análisis. El diseño mostró buen desacoplamiento entre cliente y servidores (local y remoto), lo que facilita extensibilidad y pruebas A/B de herramientas. Como mejora futura se recomienda reforzar seguridad (mTLS o tokens, CORS restrictivo, rate limiting), observabilidad (métricas por método, trazas con id), empaquetado reproducible (Docker/CI) y adopción plena de servidores MCP oficiales por STDIO cuando el entorno lo permita. En síntesis, se cumplió el objetivo de integrar, capturar y explicar la comunicación a múltiples capas, dejando una base segura y portable para añadir nuevas capacidades MCP.

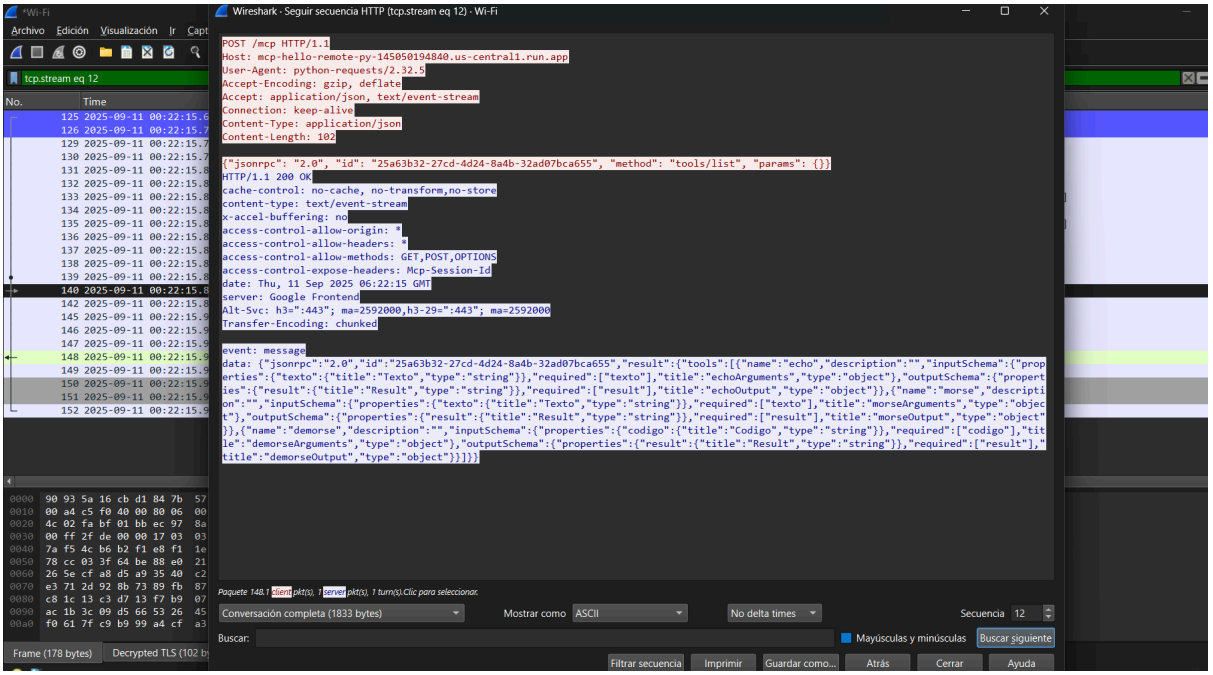
Comentario

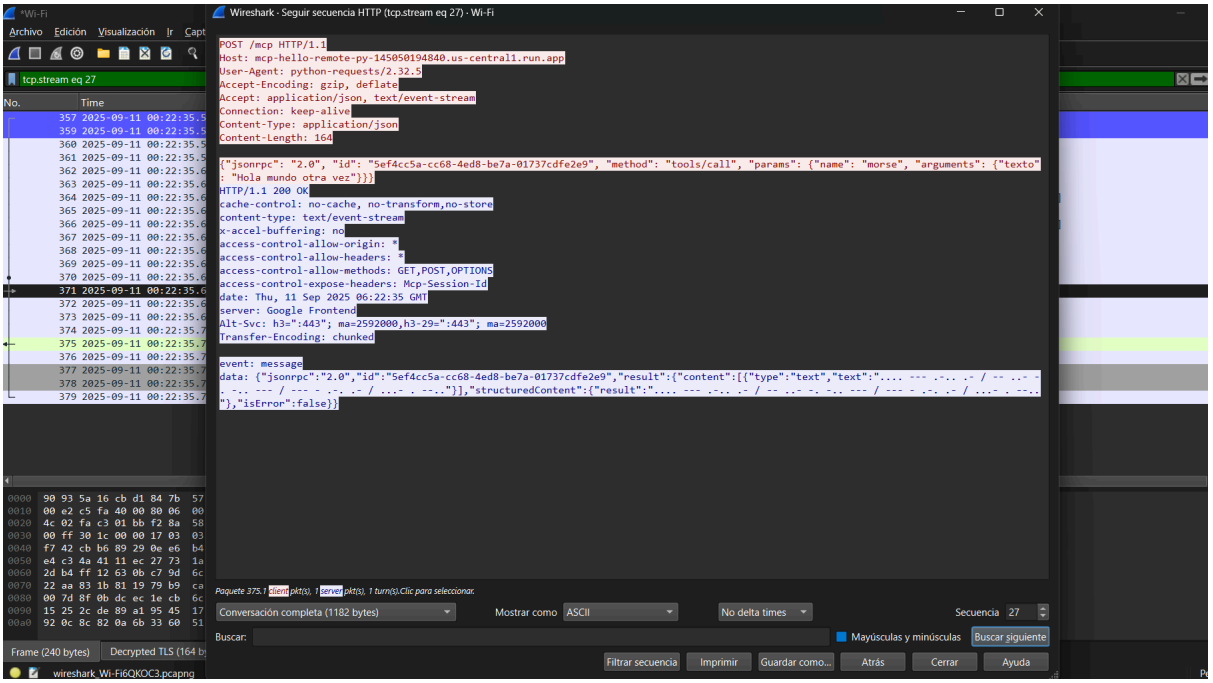
Considero que ha sido el proyecto al que más tiempo le he dedicado para poder entenderlo; deja buenas enseñanzas, pero sí resultó más complicado de lo que esperaba. Me obligó a profundizar en JSON-RPC, HTTPS/TLS, el uso de Wireshark para descifrar flujos, la configuración de entornos en Windows/venv y el manejo de llaves SSL para decrifrar tráfico. También aprendí a interpretar errores, diferenciar problemas de red vs. aplicación y a validar hipótesis con evidencia (capturas y logs) en lugar de suposiciones. Aunque la curva de aprendizaje fue alta, hoy tengo una visión más clara de cómo se conectan las capas de enlace, red, transporte y aplicación, y de cómo MCP desacopla cliente y servidor. Para futuros trabajos, planearía mejor el entorno (Docker, scripts reproducibles) e instrumentaría métricas y logs desde el inicio para reducir la fricción. En balance, el esfuerzo valió la pena y me deja una base sólida para proyectos similares.

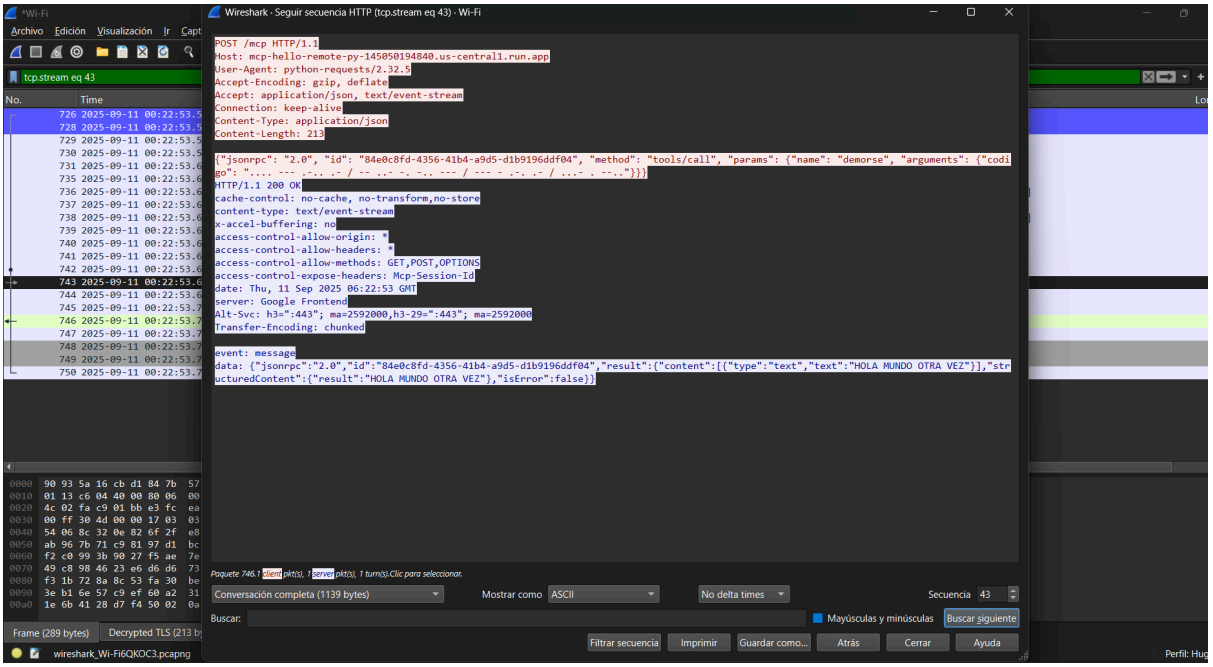
Anexo


Wireshark · Seguir secuencia HTTP (tcp.stream eq 5) · Wi-Fi
GET / HTTP/1.1
Host: mcp-hello-remote-py-145050194840.us-central1.run.app
User-Agent: python-requests/2.32.5
Accept-Encoding: gzip, deflate
Accept: */*
Connection: keep-alive
HTTP/1.1 200 OK
content-type: application/json; charset=utf-8
access-control-allow-origin: *
access-control-allow-headers: *
access-control-allow-methods: GET,POST,OPTIONS
access-control-expose-headers: Mcp-Session-Id
cache-control: no-store
date: Thu, 11 Sep 2025 06:22:14 GMT
server: Google Frontend
Alt-Svc: h3=":443"; ma=2592000,h3-29=":443"; ma=2592000
Transfer-Encoding: chunked
{"ok": true, "kind": "mcp-streamable-http", "mount": "/mcp", "tools": ["echo(texto)", "morse(texto)", "demorse(codigo)"]}
Conversación completa (702 bytes)
Mostrar como ASCII
No delta times
Secuencia 5
Buscar:
Mayúsculas y minúsculas
Buscar siguiente


Wireshark · Seguir secuencia HTTP (tcp.stream eq 11) · Wi-Fi
POST /mcp HTTP/1.1
Host: mcp-hello-remote-py-145050194840.us-central1.run.app
User-Agent: python-requests/2.32.5
Accept-Encoding: gzip, deflate
Accept: application/json, text/event-stream
Connection: keep-alive
Content-Type: application/json
Content-Length: 153
{"jsonrpc": "2.0", "id": "16210a1e-fa82-45a5-8348-6ce50c8aac1a", "method": "initialize", "params": {"protocolVersion": "2025-06-18", "capabilities": {}}}
HTTP/1.1 200 OK
cache-control: no-cache, no-transform,no-store
content-type: text/event-stream
x-accel-buffering: no
access-control-allow-origin: *
access-control-allow-headers: *
access-control-allow-methods: GET,POST,OPTIONS
access-control-expose-headers: Mcp-Session-Id
date: Thu, 11 Sep 2025 06:22:15 GMT
server: Google Frontend
Alt-Svc: h3=":443"; ma=2592000,h3-29=":443"; ma=2592000
Transfer-Encoding: chunked
event: message
data: {"jsonrpc":"2.0","id":"16210a1e-fa82-45a5-8348-6ce50c8aac1a","error":{"code":-32602,"message":"Invalid request parameters","data":""}}
Conversación completa (1015 bytes)
Mostrar como ASCII
No delta times
Secuencia 11
Buscar:
Mayúsculas y minúsculas
Buscar siguiente
Filtrar secuencia
Imprimir
Guardar como...
Atrás
Cerrar
Ayuda


Wireshark · Seguir secuencia HTTP (tcp.stream eq 12) · Wi-Fi
POST /mcp HTTP/1.1
Host: mcp-hello-remote-py-145050194840.us-central1.run.app
User-Agent: python-requests/2.32.5
Accept-Encoding: gzip, deflate
Accept: application/json, text/event-stream
Connection: keep-alive
Content-Type: application/json
Content-Length: 102
{"jsonrpc": "2.0", "id": "25a63b32-27cd-4d24-8a4b-32ad07bca655", "method": "tools/list", "params": {}}
HTTP/1.1 200 OK
cache-control: no-cache, no-transform,no-store
content-type: text/event-stream
x-accel-buffering: no
access-control-allow-origin: *
access-control-allow-headers: *
access-control-allow-methods: GET,POST,OPTIONS
access-control-expose-headers: Mcp-Session-Id
date: Thu, 11 Sep 2025 06:22:15 GMT
server: Google Frontend
Alt-Svc: h3=":443"; ma=2592000,h3-29=":443"; ma=2592000
Transfer-Encoding: chunked
event: message
data: {"jsonrpc":"2.0","id":"25a63b32-27cd-4d24-8a4b-32ad07bca655","result":{"tools":[{"name":"echo","description":"","inputSchema":{"properties":{"texto":{"title":"Texto","type":"string"}},"required":["texto"],"title":"echoArguments","type":"object"},"outputSchema":{"properties":{"result":{"title":"Result","type":"string"}},"required":["result"],"title":"echoOutput","type":"object"}},{"name":"morse","description":"","inputSchema":{"properties":{"texto":{"title":"Texto","type":"string"}},"required":["texto"],"title":"morseArguments","type":"object"},"outputSchema":{"properties":{"result":{"title":"Result","type":"string"}},"required":["result"],"title":"morseOutput","type":"object"}},{"name":"demorse","description":"","inputSchema":{"properties":{"codigo":{"title":"Codigo","type":"string"}},"required":["codigo"],"title":"demorseArguments","type":"object"},"outputSchema":{"properties":{"result":{"title":"Result","type":"string"}},"required":["result"],"title":"demorseOutput","type":"object"}}]}
Paquete 148.1 client pkt(s), 1 server pkt(s), 1 turn(s).Clic para seleccionar.
Conversación completa (1833 bytes)
Mostrar como ASCII
No delta times
Secuencia 12
Buscar:
Mayúsculas y minúsculas
Buscar siguiente
Filtrar secuencia
Imprimir
Guardar como...
Atrás
Cerrar
Ayuda


Wireshark · Seguir secuencia HTTP (tcp.stream eq 27) · Wi-Fi
POST /mcp HTTP/1.1
Host: mcp-hello-remote-py-145050194840.us-central1.run.app
User-Agent: python-requests/2.32.5
Accept-Encoding: gzip, deflate
Accept: application/json, text/event-stream
Connection: keep-alive
Content-Type: application/json
Content-Length: 164
{"jsonrpc": "2.0", "id": "5ef4cc5a-cc68-4ed8-be7a-01737cdfe2e9", "method": "tools/call", "params": {"name": "morse", "arguments": {"texto": "Hola mundo otra vez"}}}
HTTP/1.1 200 OK
cache-control: no-cache, no-transform,no-store
content-type: text/event-stream
x-accel-buffering: no
access-control-allow-origin: *
access-control-allow-headers: *
access-control-allow-methods: GET,POST,OPTIONS
access-control-expose-headers: Mcp-Session-Id
date: Thu, 11 Sep 2025 06:22:35 GMT
server: Google Frontend
Alt-Svc: h3=":443"; ma=2592000,h3-29=":443"; ma=2592000
Transfer-Encoding: chunked
event: message
data: {"jsonrpc":"2.0","id":"5ef4cc5a-cc68-4ed8-be7a-01737cdfe2e9","result":{"content":[{"type":"text","text":".... --- .-.. .- / -- ..- - . -.. --- / --- - .-. .- / ...- . --.."}],"structuredContent":{"result":".... --- .-.. .- / -- ..- -. -.. --- / --- - .-. .- / ...- . --.."},"isError":false}}
Paquete 375.1 client pkt(s), 1 server pkt(s), 1 turn(s).Clic para seleccionar.
Conversación completa (1182 bytes)
Mostrar como ASCII
No delta times
Secuencia 27
Buscar:
Mayúsculas y minúsculas
Buscar siguiente
Filtrar secuencia
Imprimir
Guardar como...
Atrás
Cerrar
Ayuda


Wireshark · Seguir secuencia HTTP (tcp.stream eq 43) · Wi-Fi
POST /mcp HTTP/1.1
Host: mcp-hello-remote-py-145050194840.us-central1.run.app
User-Agent: python-requests/2.32.5
Accept-Encoding: gzip, deflate
Accept: application/json, text/event-stream
Connection: keep-alive
Content-Type: application/json
Content-Length: 213
{"jsonrpc": "2.0", "id": "84e0c8fd-4356-41b4-a9d5-d1b9196ddf04", "method": "tools/call", "params": {"name": "demorse", "arguments": {"codi
go": "... --- ..-. .- / -- ..- -. -.. --- / --- - .-. .- / ... - --..-"}}}
HTTP/1.1 200 OK
cache-control: no-cache, no-transform,no-store
content-type: text/event-stream
x-accel-buffering: no
access-control-allow-origin: *
access-control-allow-headers: *
access-control-allow-methods: GET,POST,OPTIONS
access-control-expose-headers: Mcp-Session-Id
date: Thu, 11 Sep 2025 06:22:53 GMT
server: Google Frontend
Alt-Svc: h3=":443"; ma=2592000,h3-29=":443"; ma=2592000
Transfer-Encoding: chunked
event: message
data: {"jsonrpc":"2.0","id":"84e0c8fd-4356-41b4-a9d5-d1b9196ddf04","result":{"content":[{"type":"text","text":"HOLA MUNDO OTRA VEZ"}],"str
ucturedContent":{"result":"HOLA MUNDO OTRA VEZ"},"isError":false}}
Conversación completa (1139 bytes)
Mostrar como
ASCII
No delta times
Secuencia
43
Buscar:
Mayúsculas y minúsculas
Buscar siguiente
Filtrar secuencia
Imprimir
Guardar como...
Atrás
Cerrar
Ayuda
tcp.stream eq 43
Frame (289 bytes)
wireshark_Wi-Fi6QKOC3.pcapng