

Universidad del Valle de Guatemala
Facultad de Ingeniería
Redes



“Protocolo Existente”

Davis Roldán 22672

14 de septiembre del 2025, Guatemala de la Asunción

Especificación de los servidores MCP desarrollados (HTTP y STDIO): endpoints, métodos, parámetros

- Estándar común: Model Context Protocol (MCP) sobre JSON-RPC 2.0.
- Descubrimiento de capacidades: initialize → tools/list → tools/call.
- Contrato de entrada/salida: JSON estricto; cada tool publica su inputSchema (campos required y properties).

Servidor MCP HTTP/HTTPS (remoto)

- Transporte: HTTP/HTTPS.
- Endpoint único: POST /mcp
- Headers: Content-Type: application/json (y Accept: application/json)
- Negociación de versión: protocolVersion (ej. 2024-09 o 2025-06-18)

Initialize

```
{
  "jsonrpc": "2.0",
  "id": 1,
  "method": "initialize",
  "params": {
    "protocolVersion": "2024-09",
    "capabilities": { "tools": {}, "resources": {}, "prompts": {} },
    "clientInfo": { "name": "mcp-chatbot", "version": "0.1.0" }
  }
}
```

Respuesta típica de éxito:

```
{
  "jsonrpc": "2.0", "id": 3, "result": { "ok": true, "data": { "valid": true, "reason": "syntax+mx_ok" } } }
```

Servidores MCP STDIO (locales: Filesystem y Git)

- Transporte: tuberías estándar del proceso hijo (no hay TCP/TLS).
- Lanzamiento: comando del servidor; el cliente envía/recibe líneas JSON por STDIN/STDOUT.
- Ciclo: initialize → (algunos requieren) notifications/initialized (sin id) → tools/list → tools/call.

Handshake típico por STDIO (líneas JSON):

1. Cliente → initialize (igual que HTTP)
2. (Para Git MCP y similares) Cliente → notifications/initialized (notificación sin id; no se espera respuesta, pero desbloquea tools/list)
3. Cliente → tools/list
4. Cliente → tools/call (según inputSchema de cada tool)

análisis de enlace, red, transporte y aplicación

1. Para HTTP/HTTPS (servidor remoto):

Capa de Enlace (L2): transmisión local (Wi-Fi/Ethernet), tramas 802.11/802.3, direcciones MAC, FCS y posibles retransmisiones locales.

Capa de Red (L3): IP enruta paquetes desde la LAN del cliente hacia la IP pública del servicio (Router); NAT en el borde del cliente; TTL, posible fragmentación.

Capa de Transporte (L4): TCP establece conexión fiable (3-way handshake), control de flujo y congestión; si es HTTPS, se superpone TLS (negociación de cifrados, autenticación del servidor, canal cifrado e íntegro).

Capa de Aplicación (L7): HTTP POST a /mcp con JSON-RPC 2.0. El cliente negocia protocolVersion en initialize, descubre tools con tools/list y ejecuta tools/call con arguments que cumplen el inputSchema. El servidor valida, ejecuta y responde con result o error.

2. Para STDIO (servidores locales en el mismo host):

Capa de Enlace / Red / Transporte: no hay tránsito en red; el intercambio se realiza por STDIN/STDOUT del proceso (pipes del SO). No existe TCP/TLS ni direccionamiento IP (se evita latencia de red).

Capa de Aplicación (L7): idéntica semántica MCP/JSON-RPC que en HTTP: initialize, (opcional notifications/initialized), tools/list, tools/call. El cliente y el proceso hijo se comunican con líneas JSON. El Filesystem MCP aplica un directorio raíz permitido para seguridad.

Análisis Wireshark

The image shows a Wireshark packet capture on the Wi-Fi en0 interface. The packet list on the left shows several packets, with packet 72 selected. The packet details pane on the right shows the structure of packet 72, which is an HTTP POST request. The packet bytes pane at the bottom shows the raw data of the packet.

No.	Time	Source	Destination	Protocol	Length	Info
60	05:20:16.495716	192.168.0.2	130.211.115.4	TCP	66	52138 → 443 [ACK] Seq=1 Ack=26 Win=2048 Len=0 TSval=4113184643 TSecr=48158794
61	05:20:16.570442	130.211.115.4	192.168.0.2	TCP	66	[TCP Retransmission] 443 → 52137 [FIN, ACK] Seq=25 Ack=1 Win=1254 Len=0 TSval=48158900 TSecr=1819675
62	05:20:16.578680	192.168.0.2	130.211.115.4	TCP	66	[TCP Dup ACK 58#1] 52137 → 443 [ACK] Seq=1 Ack=26 Win=2048 Len=0 TSval=1819691948 TSecr=48158744
63	05:20:16.611516	192.168.0.2	35.244.144.25	TCP	54	51708 → 443 [ACK] Seq=1 Ack=1 Win=2048 Len=0
64	05:20:16.652184	35.244.144.25	192.168.0.2	TCP	66	[TCP ACKed unseen segment] 443 → 51708 [ACK] Seq=1 Ack=2 Win=1038 Len=0 TSval=2969194673 TSecr=41268
65	05:20:17.264155	2803:d100:9840:12a...	2803:c800:0:7d::2	DNS	115	Standard query 0xfdf A mcp-server-remote-tqnd.onrender.com
66	05:20:17.264459	2803:d100:9840:12a...	2803:c800:0:7d::2	DNS	115	Standard query 0x3e44 AAAA mcp-server-remote-tqnd.onrender.com
67	05:20:17.334382	2803:c800:0:7d::2	2803:d100:9840:12a...	DNS	250	Standard query response 0xfdf A mcp-server-remote-tqnd.onrender.com CNAME gcp-us-west1-1.origin.onrender.com
68	05:20:17.336058	2803:c800:0:7d::2	2803:d100:9840:12a...	DNS	273	Standard query response 0x3e44 AAAA mcp-server-remote-tqnd.onrender.com CNAME gcp-us-west1-1.origin.onrender.com
69	05:20:17.337466	192.168.0.2	216.24.57.7	TCP	78	52177 → 80 [SYN, ECE, CWR] Seq=0 Win=65535 Len=0 MSS=1460 WS=64 TSval=169799967 TSecr=0 SACK_PERM
70	05:20:17.387162	216.24.57.7	192.168.0.2	TCP	74	80 → 52177 [SYN, ACK, ECE] Seq=0 Ack=1 Win=65535 Len=0 MSS=1460 SACK_PERM TSval=2737486763 TSecr=169
71	05:20:17.387349	192.168.0.2	216.24.57.7	TCP	66	52177 → 80 [ACK] Seq=1 Ack=1 Win=131776 Len=0 TSval=169800017 TSecr=2737486763
72	05:20:17.387889	192.168.0.2	216.24.57.7	HTTP	483	POST /mcp HTTP/1.1, JSON (application/json)
73	05:20:17.427542	216.24.57.7	192.168.0.2	TCP	66	80 → 52177 [ACK] Seq=1 Ack=418 Win=131072 Len=0 TSval=2737486803 TSecr=169800018
74	05:20:17.694547	2803:d100:9840:12a...	2600:1901:1:1e0::	TLSv1	118	Application Data
75	05:20:17.700352	216.24.57.7	192.168.0.2	HTTP	346	HTTP/1.1 307 Temporary Redirect
76	05:20:17.700506	192.168.0.2	216.24.57.7	TCP	66	52177 → 80 [ACK] Seq=418 Ack=281 Win=131520 Len=0 TSval=1698000330 TSecr=2737487028
77	05:20:17.700938	192.168.0.2	216.24.57.7	TCP	66	52177 → 80 [FIN, ACK] Seq=418 Ack=281 Win=131520 Len=0 TSval=1698000330 TSecr=2737487028
78	05:20:17.742096	2600:1901:1:1e0::	2803:d100:9840:12a...	TCP	86	443 → 62733 [ACK] Seq=1 Ack=33 Win=1046 Len=0 TSval=557993657 TSecr=3520541182
79	05:20:17.742896	216.24.57.7	192.168.0.2	TCP	66	80 → 52177 [FIN, ACK] Seq=281 Ack=419 Win=131072 Len=0 TSval=2737487119 TSecr=1698000330
80	05:20:17.744108	192.168.0.2	216.24.57.7	TCP	66	52177 → 80 [ACK] Seq=419 Ack=282 Win=131520 Len=0 TSval=1698000373 TSecr=2737487119
81	05:20:17.772831	2600:1901:1:1e0::	2803:d100:9840:12a...	TLSv1	114	Application Data
82	05:20:17.773769	2803:d100:9840:12a...	2600:1901:1:1e0::	TLSv1	86	62733 → 443 [ACK] Seq=33 Ack=29 Win=2048 Len=0 TSval=3520541260 TSecr=557993687
83	05:20:18.033110	2803:d100:9840:12a...	2607:f80b:4008:809...	TLSv1	151	Application Data
84	05:20:18.080387	2607:f80b:4008:809...	2803:d100:9840:12a...	TLSv1	125	Application Data
85	05:20:18.080575	2803:d100:9840:12a...	2607:f80b:4008:809...	TCP	86	50794 → 443 [ACK] Seq=66 Ack=40 Win=2048 Len=0 TSval=901193561 TSecr=2078476397
86	05:20:18.235942	2803:d100:9840:12a...	2607:f80b:4008:809...	TCP	74	[TCP Keep-Alive] 50794 → 443 [ACK] Seq=65 Ack=40 Win=2048 Len=0
87	05:20:18.285305	2607:f80b:4008:809...	2803:d100:9840:12a...	TCP	86	[TCP Keep-Alive] 443 → 50794 [ACK] Seq=40 Ack=66 Win=982 Len=0 TSval=2078476605 TSecr=901193561
88	05:20:18.406660	2803:d100:9840:12a...	2600:1901:1:1e0::	TCP	74	[TCP Keep-Alive] 62733 → 443 [ACK] Seq=32 Ack=29 Win=2048 Len=0

Frame 72: 483 bytes on wire (3864 bits), 483 bytes captured (3864 bits) on interface en0, Section number: 1
Interface id: 0 (en0)
Encapsulation type: Ethernet (1)
Arrival Time: Sep 24, 2025 23:20:17.387889000 CST
UTC Arrival Time: Sep 25, 2025 05:20:17.387889000 UTC
Epoch Arrival Time: 1758777617.387889000
[Time shift for this packet: 0.000000000 seconds]
[Time delta from previous captured frame: 0.000540000 seconds]
[Time delta from previous displayed frame: 0.000540000 seconds]

Entorno observado

- Cliente (LAN): 192.168.0.2
- Servidor remoto (Render): 216.24.57.7
- Servicio: MCP sobre HTTP→HTTPS (redirect a 443/TLS)
- Interfaz capturada: Wi-Fi en0

TCP 3-way handshake (puerto 80)

- 192.168.0.2 → 216.24.57.7 :80 SYN (MSS=1460, WS=64, SACK_PERM)
- 216.24.57.7 → 192.168.0.2 :80 SYN, ACK
- 192.168.0.2 → 216.24.57.7 :80 ACK

Aplicación (HTTP en claro, puerto 80)

- Request: POST /mcp HTTP/1.1 con Content-Type: application/json ⇒ JSON-RPC (Solicitud). Es el primer intercambio MCP; en este punto, típicamente es el initialize (sincronización de protocolo).
- Response: HTTP/1.1 307 Temporary Redirect ⇒ el backend redirige a HTTPS (puerto 443).
- Nuevo TCP 3-way handshake (puerto 443)
- 192.168.0.2 → 216.24.57.7 :443 SYN (parámetros similares: MSS, WS, SACK)
- 216.24.57.7 → 192.168.0.2 :443 SYN, ACK
- 192.168.0.2 → 216.24.57.7 :443 ACK

TLS handshake (puerto 443)

- Client Hello (con SNI al host del servicio)
- Server Hello (+ negociación de cifrado)

A partir de aquí, Application Data (cifrado TLS) — el contenido HTTP/JSON-RPC ya no es visible.

Tráfico de aplicación (cifrado)

- Ráfagas Client→Server y Server→Client como TLS Application Data.
En este tramo viajan:
JSON-RPC (Solicitudes) como tools/list y tools/call (POST /mcp sobre HTTPS).
JSON-RPC (Respuestas) emparejadas por el mismo id de JSON-RPC.

No se ven los cuerpos por estar cifrados; para verlos se necesita SSLKEYLOGFILE o capturar contra HTTP local.

Pas o	Cap a	Dirección	Puert o	Detalle	Rol JSON-RPC
1	L4	192.168.0.2 → 216.24.57.7	80	TCP SYN, MSS=1460, WS=64, SACK	—
2	L4	216.24.57.7 → 192.168.0.2	80	TCP SYN, ACK	—
3	L4	192.168.0.2 → 216.24.57.7	80	TCP ACK	—
4	L7	192.168.0.2 → 216.24.57.7	80	HTTP POST /mcp Content-Type: application/json	Solicitud (p.ej. initialize)
5	L7	216.24.57.7 → 192.168.0.2	80	HTTP/1.1 307 Temporary Redirect	—
6	L4	192.168.0.2 →	443	TCP SYN	—

		216.24.57.7			
7	L4	216.24.57.7 → 192.168.0.2	443	TCP SYN, ACK	—
8	L4	192.168.0.2 → 216.24.57.7	443	TCP ACK	—
9	L7	192.168.0.2 ↔ 216.24.57.7	443	TLS handshake (ClientHello/ServerHello/Finished)	—
10	L7	192.168.0.2 ↔ 216.24.57.7	443	TLS Application Data (HTTP POST /mcp + JSON-RPC)	Solicitudes/Respuestas

Conclusión

El proyecto demuestra que MCP, sobre un mismo contrato JSON-RPC, permite operar tanto en HTTP (remoto, escalable y seguro con TLS) como en STDIO (local, rápido y con acceso controlado al entorno). Esta convergencia reduce el acoplamiento: el cliente negocia initialize, descubre tools/list y ejecuta tools/call sin importar el transporte. Además, los normalizadores y guardas en el cliente (validación de argumentos, mapeo server→tool y notificación initialized donde aplica) vuelven la ejecución robusta y predecible. En seguridad, HTTPS y la restricción de directorios en Filesystem MCP ofrecen un equilibrio práctico entre exposición y protección. En conjunto, la arquitectura es simple de extender (añadir nuevas tools), fácil de mantener y lista para endurecer con observabilidad (métricas, trazabilidad) y controles de acceso más finos según crezcan las necesidades.