

**Proyecto: Vehículo Autónomo de Telemetría**

Alejandro Arteaga  
Mayerly Alejandra Suarez  
Paula Llanos

Escuela Ciencias Aplicadas e Ingeniería, Universidad EAFIT  
Telemática  
Profesor: Alber Oswaldo Montoya Benitez

Medellín, Colombia  
4 de octubre de 2025

## Índice

<b>1. Introducción.....</b>	<b>3</b>
Objetivo del protocolo:	
Definir un mecanismo de comunicación claro, eficiente y extensible que permita:.....	3
<b>2. Componentes:.....</b>	<b>4</b>
Servidor(C).....	4
Clientes (Java/Python).....	4
<b>3. Visión General del Protocolo.....</b>	<b>4</b>
Roles:.....	5
Capacidades del protocolo:.....	5
<b>4. Formato de los mensajes.....</b>	<b>5</b>
<b>5. Especificación del Servicio.....</b>	<b>5</b>
5.1 (Cliente → Servidor).....	6
5.2 (Servidor → Cliente).....	6
<b>6. Reglas del servidor para los controles del administrador.....</b>	<b>7</b>
6.1 Reglas para comandos de control.....	7
6.2 Manejo de errores.....	8
6.3 Formato de Mensajes y encabezado.....	8
6.4 Reglas o consideraciones extras.....	8
<b>7. Ejemplos de Implementación.....</b>	<b>8</b>
<b>8. Anexos.....</b>	<b>10</b>
8.1 Tabla de comandos.....	10
8.2 Campos de telemetría.....	10
9. Referencias.....	11

# 1. Introducción

El presente documento describe la especificación del protocolo de comunicación diseñado para el sistema **Vehículo Autónomo – Telemetría**, bajo una arquitectura cliente–servidor. Este sistema tiene como propósito permitir que múltiples clientes se conecten a un servidor central, reciban periódicamente datos de telemetría (velocidad, nivel de batería, temperatura y dirección del vehículo) y, en el caso de usuarios con privilegios de administrador, puedan enviar comandos de control para modificar el comportamiento del vehículo.

El protocolo, denominado **TLP/1.0**, se fundamenta en un canal confiable orientado a conexión (TCP/IP) y emplea mensajes de texto simples para garantizar la compatibilidad con clientes ligeros y facilitar su implementación.

## Objetivo del protocolo:

Definir un mecanismo de comunicación claro, eficiente y extensible que permita:

1. La transmisión periódica y estandarizada de telemetría del vehículo hacia los clientes.
2. La autenticación de usuarios con diferentes roles (observador y administrador).
3. El envío de comandos de control desde administradores hacia el vehículo, con validación de reglas de seguridad y operación.
4. La interoperabilidad entre distintos clientes (ej. implementaciones en Java o Python) sin ambigüedades en el formato de los mensajes.

Con este protocolo se busca establecer una base formal que asegure el monitoreo confiable del vehículo autónomo y un control remoto seguro, ofreciendo un diseño simple pero adaptable a futuras versiones o ampliaciones funcionales

## 2. Componentes:

El sistema Vehículo Autónomo – Telemetría está compuesto por dos entidades principales que interactúan mediante el protocolo TLP/1.0: el servidor y los clientes. Cada uno cumple funciones específicas dentro de la arquitectura cliente–servidor.

### Servidor(C)

El servidor constituye el núcleo del sistema. Su función principal es aceptar conexiones entrantes de múltiples clientes, mantener el registro de usuarios conectados y gestionar la comunicación. Además, es responsable de difundir periódicamente mensajes de telemetría con sus clientes, procesar comandos de clientes administradores, aplicar reglas de validación a las condiciones del vehículo y registrar eventos que afecten al servidor.

## Cientes (Java/Python)

Los clientes son aplicaciones externas que establecen conexión con el servidor utilizando TCP/IP, para el proyecto se contemplan dos tipos principales de clientes que fueron implementados tanto en Python como Java:

- Clientes observadores (ROLE\_OBSERVER): pueden suscribirse a la telemetría para recibir periódicamente información del vehículo. Sus comandos están limitados a consultas básicas como HELLO, SUBSCRIBE o LIST USERS.
- Clientes administradores (ROLE\_ADMIN): poseen todos los privilegios del observador y, adicionalmente, tienen la capacidad de enviar comandos de control (SPEED UP, SLOW DOWN, TURN LEFT, TURN RIGHT).

## 3. Visión General del Protocolo

El protocolo de comunicación definido para el proyecto se denomina **TLP/1.0 (Telematics Lightweight Protocol, versión 1.0)**. Se trata de un protocolo orientado a texto, diseñado para operar sobre conexiones confiables establecidas mediante **TCP/IP**. Su objetivo es proporcionar un canal de comunicación claro, ligero y extensible, que facilite tanto la transmisión periódica de telemetría como el envío de comandos de control.

El protocolo se implementa bajo un modelo **cliente-servidor**. El servidor mantiene un socket abierto en un puerto predefinido y acepta múltiples conexiones concurrentes. Los clientes, una vez conectados, pueden autenticarse y suscribirse al flujo de telemetría. En caso de contar con privilegios administrativos, pueden además enviar órdenes de control que modifican el estado del vehículo autónomo.

### Roles:

- Observer (**ROLE\_OBSERVER**): los clientes observadores pueden conectarse, realizar un saludo inicial (HELLO), suscribirse a la telemetría (SUBSCRIBE)
- Admin (**ROLE\_ADMIN**): los clientes administradores heredan las capacidades del observador y, adicionalmente, pueden autenticarse mediante un token (AUTH ADMIN ) para ejecutar comandos de control (COMMAND <acción>) y listar usuarios (LIST\_USERS).

### Capacidades del protocolo:

- **Transmisión periódica de telemetría:** cada cliente suscrito recibe, en intervalos configurados (10 segundos), información del vehículo en formato clave-valor (velocidad, batería, temperatura, dirección y marca temporal).
- **Control remoto mediante comandos:** los administradores pueden modificar parámetros de operación del vehículo (ej. acelerar, frenar, girar).

- **Enumeración de usuarios conectados:** a través del comando LIST USERS se obtiene un listado detallado de los clientes activos, incluyendo dirección IP, rol y tiempo de conexión.
- **Gestión de errores y validaciones:** el protocolo contempla respuestas específicas ante intentos inválidos, como fallas de autenticación, comandos no reconocidos o restricciones operativas (ej. bloqueo por sobrecalentamiento).

## 4. Formato de los mensajes

Los mensajes del protocolo se definen como líneas de texto terminadas en el carácter de nueva línea (\n). El primer token de cada línea corresponde al **tipo de mensaje** (ej. HELLO, SUBSCRIBE, TELEMETRY, COMMAND, USER...), mientras que los tokens siguientes se utilizan para argumentos o pares **clave=valor**. No se emplean delimitadores adicionales ni longitudes prefijadas, lo que simplifica la implementación y garantiza la compatibilidad entre clientes. Los mensajes siempre son tipo UTF-8/ASCII, como mencionamos, terminan en \n. La longitud máxima por línea es de 1024 caracteres.

## 5. Especificación del Servicio

Definimos un conjunto de acciones que permiten a los clientes interactuar con el servidor. Estos servicios se dividen en dos categorías: **solicitudes enviadas por los clientes** y **respuestas emitidas por el servidor**

### 5.1 (Cliente → Servidor)

Las operaciones habilitadas para los clientes son:

- **HELLO:** Inicia un saludo básico entre el cliente y el servidor.
  - Solicitud: HELLO
  - Respuesta: OK HELLO
- **SUBSCRIBE:** Permite a un cliente suscribirse para recibir telemetría periódica.
  - Solicitud: SUBSCRIBE
  - Respuesta: OK SUBSCRIBED
- **AUTH ADMIN:** Autentica al cliente como administrador mediante un token.
  - Solicitud: AUTH ADMIN <token>
  - Respuestas:
    - OK AUTH ADMIN (éxito)
    - ERROR AUTH bad\_token (fallo de autenticación)
- **LIST USERS(Solo admin):** Devuelve el listado de usuarios actualmente conectados al servidor.
  - Solicitud: LIST USERS
  - Respuesta:
    - USERS <n> (donde *n* es el número de usuarios conectados).

- Posteriormente, una línea por usuario con el formato:  
USER <ip>:<port> <ROL> <connected\_at\_unix>
- **COMMAND <acción>** (*admin*): Permite enviar instrucciones de control al vehículo autónomo.
  - Solicitud: COMMAND <acción>
  - Acciones válidas: SPEED UP, SLOW DOWN, TURN LEFT, TURN RIGHT
  - Respuesta:
    - OK EXECUTED (ejecución exitosa)
    - ERROR ... (cuando se incumple alguna regla de validación).
- **QUIT**: el servidor cierra la conexión.
  - Solicitud: QUIT
  - Respuesta: OK BYE

## 5.2 (Servidor → Cliente)

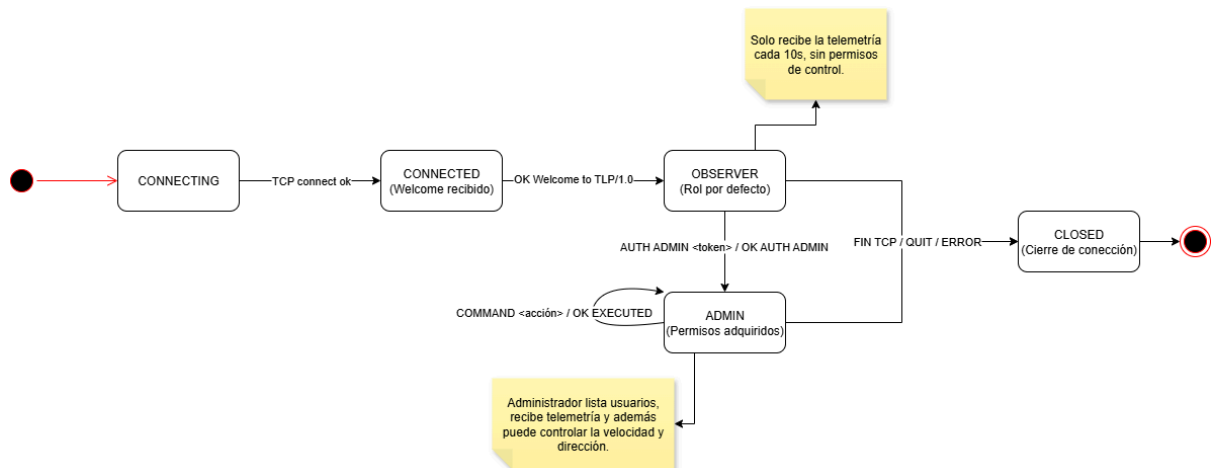
El servidor genera diferentes tipos de mensajes hacia los clientes, entre los que destacan:

- **TELEMETRY**  
Contiene la información periódica del estado del vehículo.
  - Formato:
    - TELEMETRY speed=<kmh> battery=<pct> temp=<C>  
dir=<LEFT|RIGHT|STRAIGHT> ts=<epoch>
- **OK**: Confirma la correcta ejecución de una instrucción.
- **ERROR**: Informa al cliente sobre un fallo en la solicitud. Ejemplos:
  - ERROR AUTH bad\_token
  - ERROR PERM admin\_required
  - ERROR CMD unknown\_command
  - ERROR REJECTED overheat
- **QUIT**: el servidor cierra la conexión TCP.
  - Solicitud: QUIT
  - Respuesta: OK BYE

### Ejemplo de flujo básico

1. El cliente se conecta al servidor y recibe:  
OK Welcome to TLP/1.0
2. El cliente envía: (por default se suscribe al conectarse)  
SUBSCRIBE  
→ Servidor responde: OK SUBSCRIBED
3. Cada 10 segundos el servidor envía:  
TELEMETRY speed=10 battery=95 temp=20 dir=STRAIGHT ts=1759525432

El flujo general de cliente lo podemos representar gráficamente:



## 6. Reglas del servidor para los controles del administrador

### 6.1 Reglas para comandos de control

El servidor realiza algunas validaciones antes de aceptar alguna orden del cliente con poderes de administrador.

- SPEED UP se rechaza si:
  - La temperatura excede el umbral de seguridad ( $\geq 92^{\circ}\text{C}$ ).
  - El nivel de batería es menor al 10 %.
- SLOW DOWN se rechaza si la velocidad ya es igual a 0 km/h.
- TURN LEFT o TURN RIGHT establecen la dirección solo durante dos ciclos de telemetría, tras lo cual el estado vuelve a STRAIGHT.

Se deben tener en cuenta también los siguientes criterios:

- Para SPEED UP se tiene que la velocidad aumenta de 5 km/h llegando a una velocidad máxima de 120km/h, luego al aumentar la velocidad el porcentaje de batería disminuye en 1% siempre y cuando la batería sea mayor a 0, y a su vez la temperatura aumentará de a  $3^{\circ}\text{C}$  (máxima de  $95^{\circ}\text{C}$ ).
- Para el comando SLOW DOWN la velocidad disminuye de 5 km/h con una velocidad mínima de 0 km/h, si al ejecutar el comando la temperatura es mayor a  $20^{\circ}\text{C}$ , esta disminuye  $1^{\circ}\text{C}$ .

### 6.2 Manejo de errores

El protocolo define mensajes de error estandarizados:

- ERROR AUTH bad\_token — token inválido en autenticación.
- ERROR PERM admin\_required — intento de comando sin rol administrador.
- ERROR CMD unknown\_command — comando no reconocido.
- ERROR REJECTED <razón> — rechazo por regla de negocio (ej. *overheat*, *low\_battery*, *already\_stopped*).
- ERROR BAD\_REQUEST syntax — sintaxis inválida en el mensaje.

### 6.3 Formato de Mensajes y encabezado

- **Codificación:** ASCII/UTF-8 sin NUL; el servidor **ignora** \r y termina en \n .
- **Una línea = un mensaje.** Límite: MAX\_LINE = 1024 caracteres.
- **Sintaxis:** comandos con argumentos separados por espacio; telemetría en formato clave=valor.
- El **primer token** (HELLO, SUBSCRIBE, AUTH, LIST, COMMAND, OK, ERROR, TELEMETRY, USERS, USER) funciona como **campo Type**.
- El **segundo token** (si existe) actúa como **subtipo/familia** (ADMIN, PERM, CMD, BAD\_REQUEST, REJECTED).

### 6.4 Reglas o consideraciones extras

- **Límites:** MAX\_LINE=1024, MAX\_CLIENTS=20.
- **Concurrencia:** un hilo por cliente + 1 hilo de telemetría (broadcast). Locks separados para g\_clients y g\_state.
- **Caídas de clientes:** al fallar send\_line en broadcast o recv\_line en el hilo del cliente, el servidor llama remove\_client(fd) y libera recursos.

## 7. Ejemplos de Implementación

A continuación presentamos algunos casos prácticos de ejecución y uso del servidor desarrollado, de esta manera aclaramos su funcionamiento y los distintos escenarios de interacción entre clientes y el servidor. Estos ejemplos permiten observar la estructura de los mensajes, la validación de comandos y el manejo de errores bajo las condiciones establecidas.

- **Ejecución del servidor**

```
$ make
```

```
$ make run
```

```
[2025-10-05 12:00:00] SERVER START port=8080 (socket type:
SOCK_STREAM/TCP)
```



El servidor se compila utilizando **gcc** con soporte para hilos (**-pthread**) y se ejecuta especificando el puerto y el archivo de logs. La salida indica el inicio correcto del servicio.

- **Prueba manual con netcat (Observer)**

```
$ nc 127.0.0.1 8080
OK Welcome to TLP/1.0
TELEMETRY speed=10 battery=100 temp=15 dir=STRAIGHT ts=1759525432
```

En este caso, un cliente en rol de *Observer* se conecta al servidor mediante **netcat**. Se recibe un mensaje de bienvenida seguido de un flujo de telemetría que incluye velocidad, batería, temperatura, dirección y marca de tiempo.

- **Autenticación y comando (ADMIN)**

```
AUTH ADMIN SECRETO_2025
OK AUTH ADMIN
COMMAND SPEED UP
OK EXECUTED
TELEMETRY speed=15 battery=99 temp=18 dir=STRAIGHT ts=1759525452
```

Aquí, un cliente solicita autenticarse como *ADMIN* con la clave correspondiente. Una vez validada, se envía un comando para aumentar la velocidad. El servidor confirma la ejecución y emite telemetría actualizada reflejando el cambio.

- **Manejo de errores (Aceleración con temperatura alta)**

```
COMMAND SPEED UP
ERROR REJECTED overheat
```

En este escenario, el servidor rechaza un comando debido a condiciones inseguras, como el sobrecalentamiento. Esto demuestra la capacidad del sistema para aplicar restricciones y salvaguardas.

- **Listado de usuarios conectados**

```
LIST USERS
USERS 2
USER 127.0.0.1:49522 ADMIN 1759525400
USER 127.0.0.1:49530 OBSERVER 1759525415
```

La consulta de usuarios devuelve un listado con el número de clientes conectados, sus roles y las marcas de tiempo de conexión. Esto permite la gestión y supervisión activa de la sesión.

## 8. Anexos

### 8.1 Tabla de comandos

Solicitud	Respuesta esperada (dependiendo del caso)
HELLO	OK HELLO
SUBSCRIBE	OK SUBSCRIBED
AUTH ADMIN <token>	OK AUTH ADMIN   ERROR AUTH bad_token
LIST USERS(solo admin)	USERS n + USER ...
COMMAND <acción>	OK EXECUTED   ERROR PERM admin_required   ERROR CMD unknown_command   ERROR REJECTED <razón>
QUIT	OK BYE

### 8.2 Campos de telemetría

Campo	Descripción
speed	Velocidad (km/h)
battery	Batería (%)
temp	Temperatura (°C)
dir	Dirección (LEFT/RIGHT/STRAIGHT)
ts	Marca de tiempo UNIX (s)

## 9. Referencias

- Oracle. (s.f.). *Trail: Creating a GUI With JFC/Swing*. Java Tutorials.  
<https://docs.oracle.com/javase/tutorial/uiswing/examples/concurrency/index.html>

- Python Software Foundation. (2025). *tkinter* — *Python interface to Tcl/Tk*. Python.  
<https://docs.python.org/3/library/tkinter.html>
- <https://github.com/biraj21/tcp-server/>