**Introduction**

En este laboratorio se desarrolló un sistema cliente-servidor para la transmisión confiable de mensajes a través de sockets. Se utilizó el **código Hamming (7,4)** como técnica de detección y corrección de errores.  
El proyecto se implementó en **C (cliente)** y **Python (servidor/receptor)**, con el objetivo de simular entornos de comunicación y validar la eficiencia del algoritmo frente a posibles errores en la transmisión.

**Objetivos**

* **General:**

Implementar un esquema de comunicación cliente-servidor que utilice Hamming para asegurar la integridad de los mensajes transmitidos.

* **Específicos:**

1. Desarrollar un cliente en C capaz de codificar mensajes con Hamming y enviarlos por sockets.
2. Implementar un servidor en Python que reciba los mensajes, decodifique con Hamming y registre los resultados en un archivo CSV.
3. Analizar la tasa de éxito de la transmisión, errores corregidos y mensajes inválidos.
4. Visualizar los resultados mediante gráficas para comparar el rendimiento.

**Metodología**

* **Arquitectura del sistema**
* **Cliente (C):**
  + Codifica mensajes con Hamming.
  + Envía los mensajes mediante sockets a la dirección IP del servidor.
* **Servidor (Python):**
  + Recibe los mensajes.
  + Decodifica aplicando Hamming.
  + Registra resultados en un archivo results.csv.
* **Herramientas utilizadas**
* Lenguaje C (GCC en Windows con MinGW).
* Python 3.12 con librerías: socket, csv, pandas, matplotlib.
* Git/GitHub para control de versiones.
* PowerShell para compilación y ejecución.
* **Flujo de ejecución**

1. Se compila el cliente con:

gcc -o hamming.exe hamming\_encryptor.c net\_client.c includes.c -lws2\_32

1. Se inicia el servidor

python server.py

1. El cliente envía **1000 mensajes** al servidor.
2. El servidor guarda los resultados en results.csv.
3. Se ejecuta el script de visualización:

python plot\_results.py

**Resultados**

* El cliente logró enviar mensajes correctamente utilizando Hamming.
* El servidor procesó los mensajes, pero en pruebas iniciales solo se registraron **7 outputs válidos** debido a problemas de formato (saltos de línea y compatibilidad).
* Posteriormente se optimizó el código para:
  + Ignorar saltos de línea en el receptor.
  + Aceptar más mensajes consecutivos.
* Con las mejoras, el sistema comenzó a registrar un número mayor de mensajes recibidos correctamente.

**Conclusiones**

1. Se logró implementar un esquema cliente-servidor que utiliza Hamming para detección y corrección de errores.
2. El uso de Python para el servidor facilitó la manipulación de resultados y la visualización de datos.
3. Los problemas iniciales de transmisión evidencian la importancia de manejar adecuadamente los formatos de envío/recepción de datos (e.g., saltos de línea).
4. El código Hamming demostró ser útil para detectar y corregir errores de un solo bit, mejorando la confiabilidad en la comunicación.
5. Es posible optimizar aún más la implementación para mejorar el rendimiento en transmisiones grandes (ejemplo: 1000 mensajes).