Laboratorio 2

Autores:

Kristopher Javier Alvarado López	| Carné 21188
Mario Antonio Guerra Morales	| Carné 21008
David Jonathan Aragón Vásquez	| Carné 21053

Curso:
Redes
Catedrático:
Miguel Novella Linares
Sección:
10

Universidad del Valle de Guatemala
11 calle 15-79 Zona 15 Vista Hermosa lll
Guatemala, C. A.
Facultad de Ingeniería

Laboratorio 2 - Esquemas de Detección y Corrección

Repositorio utilizado en la práctica

https://github.com/AragonD19/Lab2Redes

Descripción de la práctica y metodología utilizada

El siguiente laboratorio consiste en crear y programar algoritmos que se dediquen a la detección y corrección de errores en transmisión de mensajes entre programas de distinto lenguaje en formato ASCII. Se implementaron tres algoritmos en este laboratorio. Dos de corrección de errores, siendo estos Hamming 7:4 y el algoritmo de Viterbi y Códigos Convolucionales con una tasa de 2:1 y uno de detección de errores, el cual se trata del algoritmo CRC-32.

Para realizar esta implementación, en la primera parte de este laboratorio se realizó el funcionamiento de tres programas, uno por cada algoritmo propuesto. Los cuales consistían en un emisor, programado en C++, el cual recibiría el input o mensaje del usuario y sería codificado, para posteriormente, en un receptor programado en Python, recibir estos mensajes codificados y decodificarlos para mostrar el output o el mensaje decodificado al usuario. Luego, en esta segunda parte, se unificaron los tres algoritmos para implementar un único emisor en C++ y un único receptor en Python. De esta manera, el usuario podrá decidir qué algoritmo utilizar y poder enviar su mensaje con normalidad, el cual será recibido por el mismo receptor, indicando el algoritmo utilizado, el mensaje que el emisor envió y si este contó con errores o no.

Los dos programas cuentan con una capa de aplicación, la cual permite solicitar el mensaje en el emisor y mostrar el mensaje en el receptor. Una capa de presentación, la cual codifica el mensaje descrito por el emisor en ASCII binario, y en el receptor, ésta indica a la capa de aplicación si al decodificar el mensaje se presentaron errores y las respectivas correcciones. Una capa de enlace, en la cual se encuentran los algoritmos implementados en la primera parte, y resuelven la codificación y decodificación del mensaje para el emisor y receptor respectivamente. Por último, el emisor cuenta únicamente con una "capa de ruido" la cual hace que los bits puedan verse afectados con cierta probabilidad definida en el código, además, ambos programas cuentan con una capa de transmisión. Esta capa hace que, por medio de sockets, el emisor sea capaz de enviarle el algoritmo utilizado y el mensaje codificado al receptor, esto para llevar a cabo su labor en las capas anteriormente mencionadas.

Por último, se hicieron pruebas en base a la cantidad de caracteres y la probabilidad de ruido, además de contar con un README.md que cuenta con toda la información necesaria para compilar y ejecutar tanto el emisor.cpp, el receptor.py y el pruebas.cpp para crear las pruebas.

Resultados

Ejecución de algoritmos en emisor.cpp y receptor.py:

```
C:\Users\mague\OneDrive\Documentos\Lab2Redes>g++ emisor.cpp -o emisor.exe -l
ws2_32

C:\Users\mague\OneDrive\Documentos\Lab2Redes>emisor.exe
```

```
C:\Users\mague\OneDrive\Documentos\Lab2Redes>python receptor.py
Esperando conexión...
Algoritmo utilizado: Hamming
Mensaje codificado recibido.
```

```
C:\Users\mague\OneDrive\Documentos\Lab2Redes>emisor.exe
Introduce el mensaje en texto: hola
Selecciona el algoritmo a utilizar:
1. Hamming
2. Viterbi
3. CRC-32
1
Mensaje enviado
```

```
Error detected at bit position: 30
Error corrected.
Errors detected and corrected.
Mensaje decodificado en texto: hola
Esperando conexión...
```

```
C:\Users\mague\OneDrive\Documentos\Lab2Redes>emisor.exe
Introduce el mensaje en texto: Esto es una prueba
Selecciona el algoritmo a utilizar:
1. Hamming
2. Viterbi
3. CRC-32
2
Mensaje enviado
```

```
Se detectaron errores. Mensaje corregido:

Mensaje decodificado en texto: Esto es una prueba
Esperando conexión...
```

```
C:\Users\mague\OneDrive\Documentos\Lab2Redes>emisor.exe
Introduce el mensaje en texto: waos
Selecciona el algoritmo a utilizar:
1. Hamming
2. Viterbi
3. CRC-32
3
Mensaje enviado
```

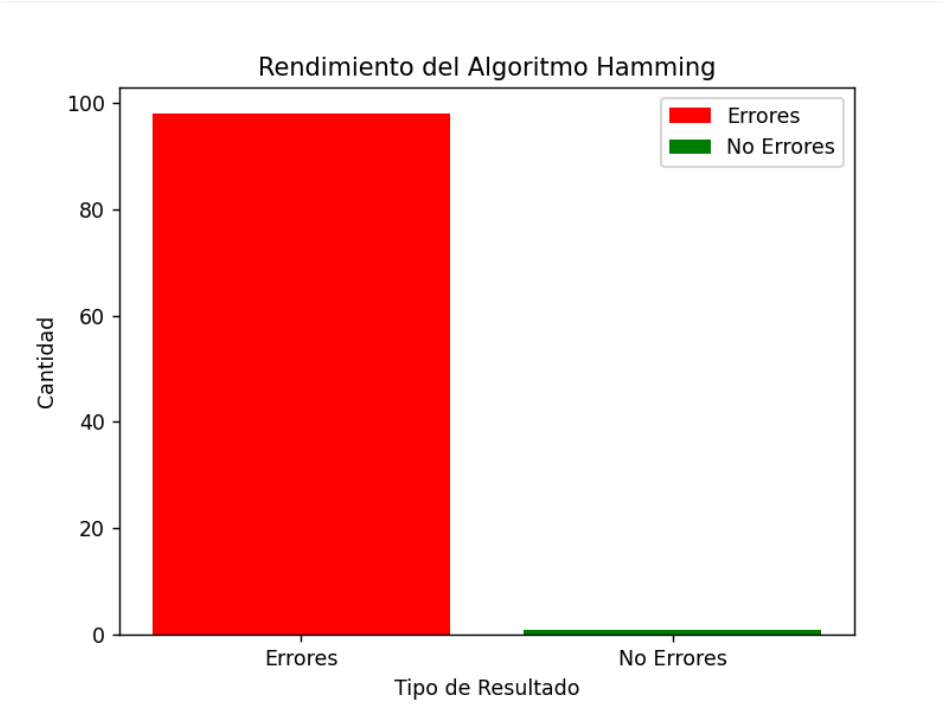
```
C:\Users\mague\OneDrive\Documentos\Lab2Redes>python receptor.py
Esperando conexión...
Algoritmo utilizado: CRC-32
Mensaje codificado recibido.

No se detectaron errores. Mensaje original: waos
Esperando conexión...
```

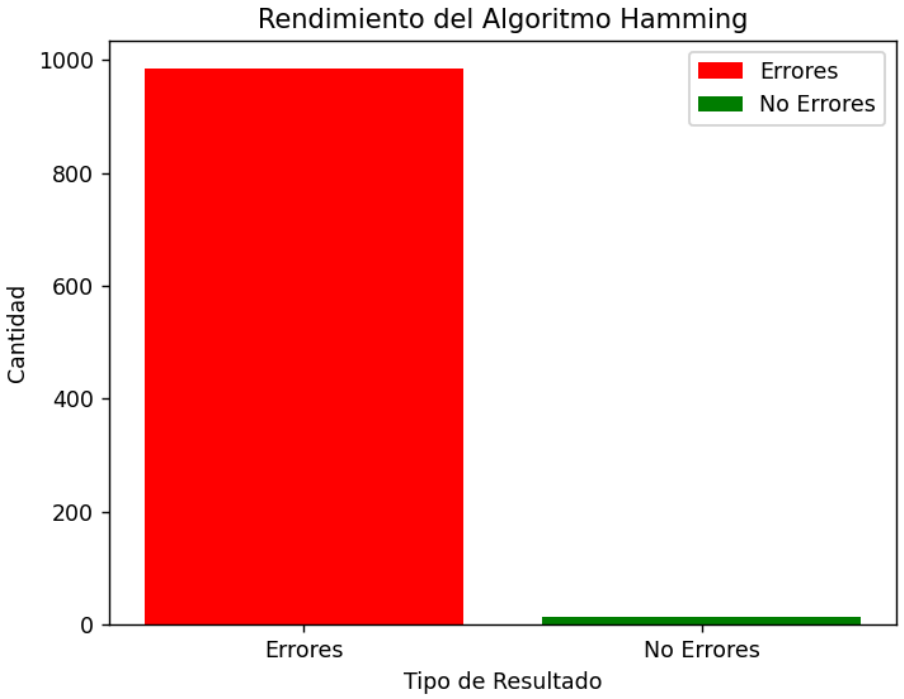
Pruebas automatizadas:

Hamming:

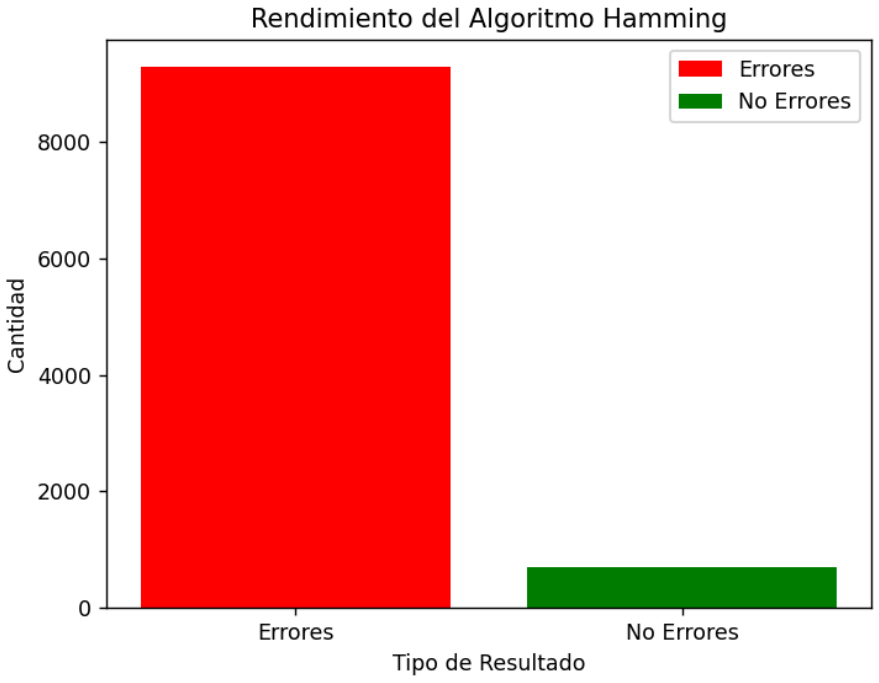
- Cantidad de pruebas: 100, con palabras de 5 a 7 caracteres y con probabilidad de ruido del 40%



- Cantidad de pruebas: 1000, con palabras de entre 5 y 7 caracteres de longitud y con una probabilidad de ruido de 25%

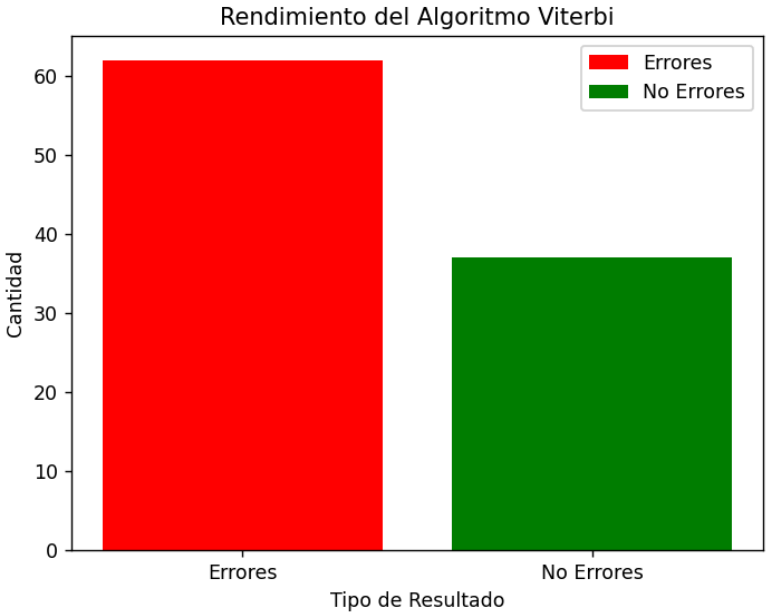


- Cantidad de pruebas: 10000, con palabras entre 5 y 7 caracteres de longitud y con una probabilidad de ruido de 15%

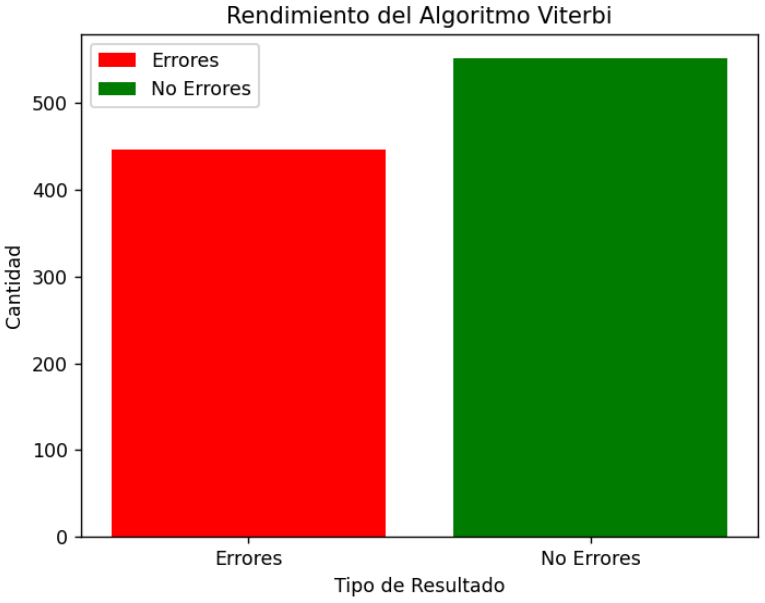


Viterbi:

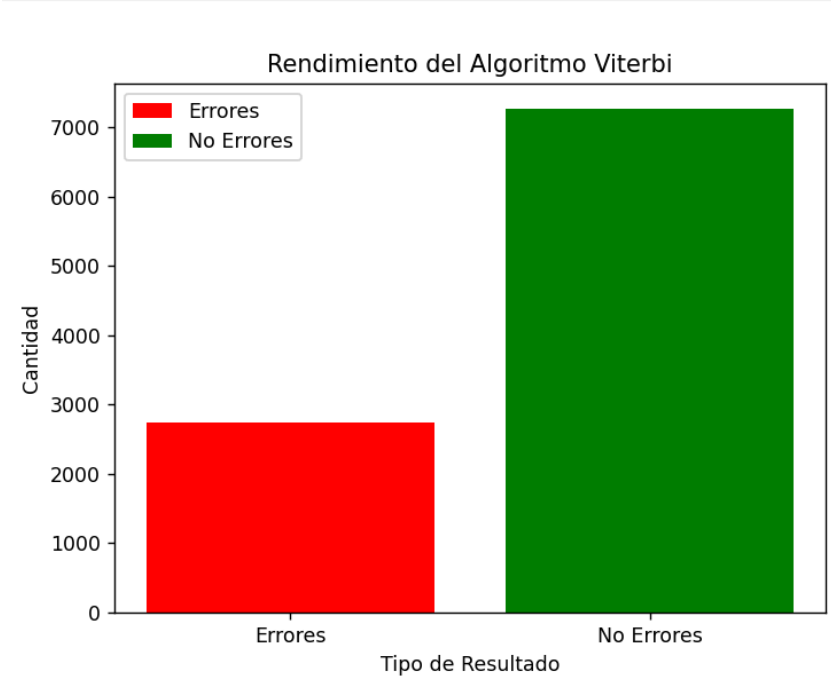
- Cantidad de pruebas: 100, con palabras de 5 a 7 caracteres y con probabilidad de ruido del 40%



- Cantidad de pruebas: 1000, con palabras de entre 5 y 7 caracteres de longitud y con una probabilidad de ruido de 25%

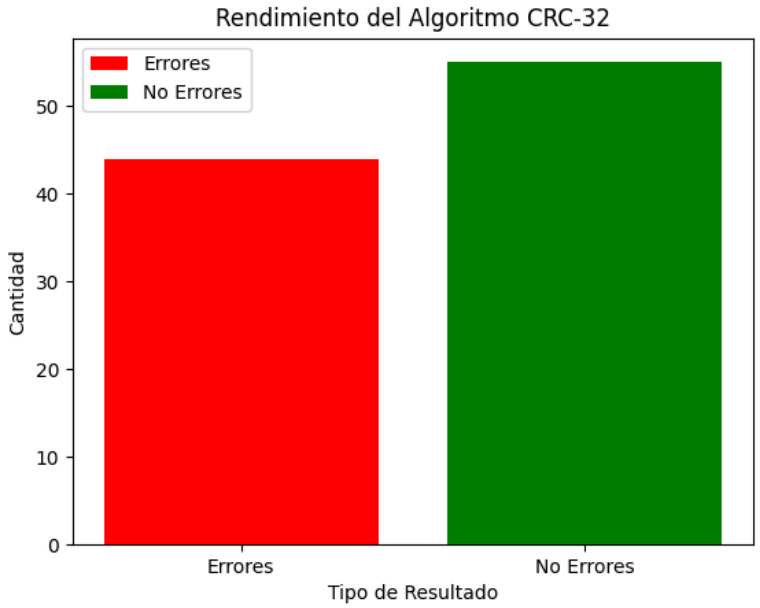


- Cantidad de pruebas: 10000, con palabras entre 5 y 7 caracteres de longitud y con una probabilidad de ruido de 15%

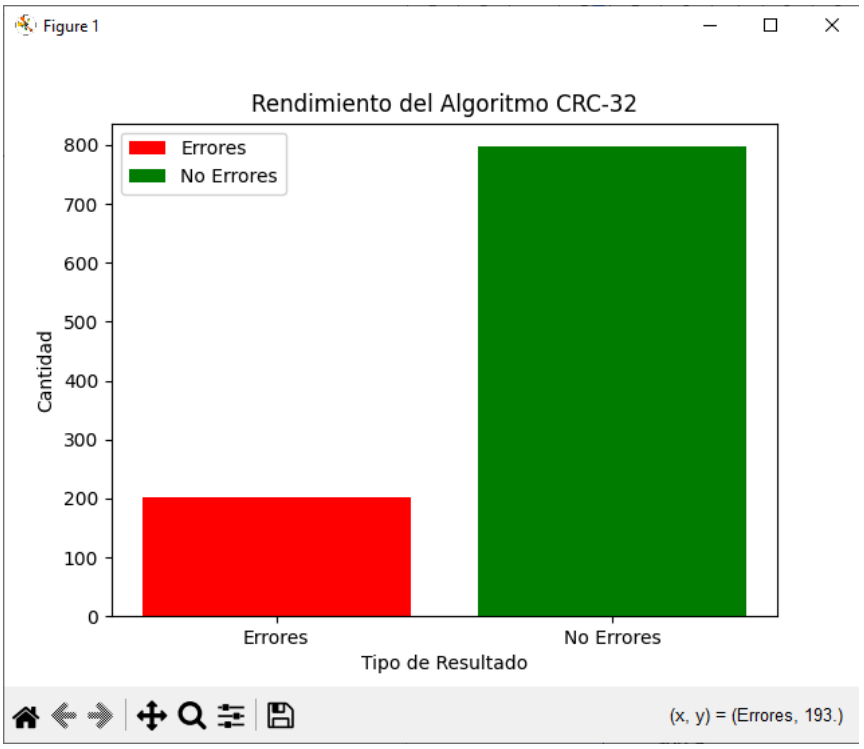


CRC32:

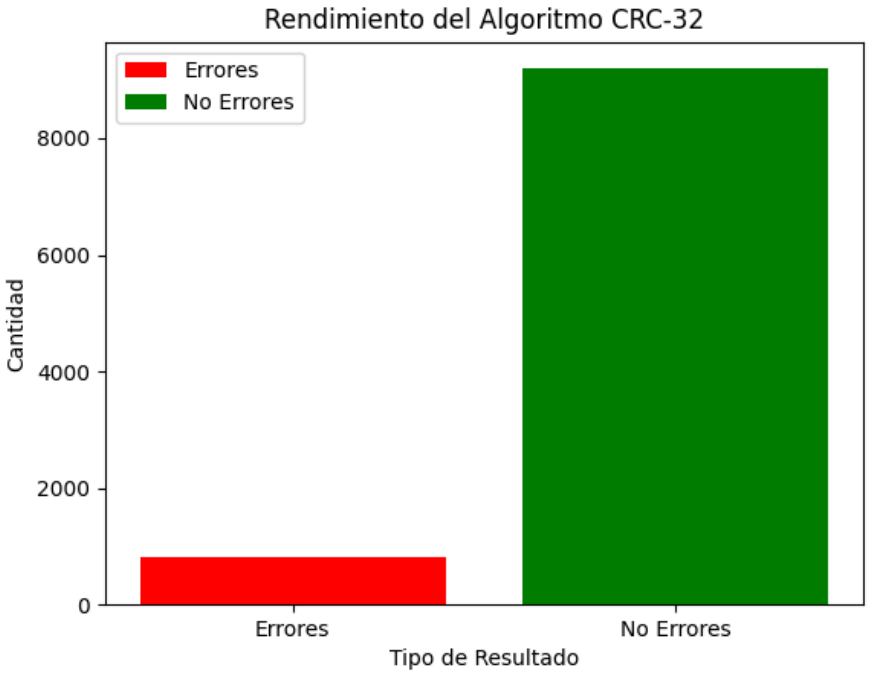
- Cantidad de pruebas: 100, con palabras de entre 10 y 30 caracteres de longitud y con una probabilidad de ruido de 40%



- Cantidad de pruebas: 1000, con palabras de entre 10 y 30 caracteres de longitud y con una probabilidad de ruido de 25%



- Cantidad de pruebas: 10000, con palabras entre 10 y 30 caracteres de longitud y con una probabilidad de ruido de 15%



Discusión de resultados

Como se puede observar, el algoritmo que tuvo un mejor funcionamiento fue el algoritmo CRC-32. Esto puede deberse principalmente a la naturaleza del algoritmo a únicamente detectar errores, ya que, tanto Hamming como Viterbi se enfocan especialmente en la corrección de errores, y Hamming únicamente es capaz de corregir errores que posean un bit modificado, mas no dos o más.Mientras que Viterbi depende más de los patrones específicos que requiera el mensaje para comprobar si, a pesar de tener errores es capaz de obtener el mensaje corregido correctamente.

En segundo lugar va el algoritmo de Viterbi, ya que este, a pesar de tener más tasa de ruido y menor cantidad de pruebas suele estar más nivelada la cantidad de errores y de no errores en comparación a los otros algoritmos, mientras que con mayor cantidad de pruebas la tendencia suele ser más favorable a no tener tantos errores.

Mientras que Hamming es el que tuvo un funcionamiento menos acertado. Puede deberse a la naturaleza del algoritmo ya anteriormente mencionada, así como también la variación de los bits de paridad al ser una implementación (7, 4), y variar entre 5 a 7 bits como se indicó en el apartado de resultados.

El algoritmo de Viterbi por otro lado, es el algoritmo capaz de aceptar mayores tasas de errores. Esto es así gracias a que Viterbi posee tasas de bits ya sea 2:1 o 3:1, lo cual indica una mayor probabilidad en producir cadenas binarias más extensas y de acuerdo a la implementación, una mayor probabilidad de incluir ruido en más de un bit de la cadena binaria. CRC-32 no contó con bastantes errores, lo cual también lo hace un algoritmo bastante resistente a las tasas de errores y de ruido.

Por último, es de considerar que como tal es importante considerar ambos tipos de algoritmos ya que se pueden requerir para resolver problemas de acuerdo al contexto que se puede dar. Sin embargo, es mejor utilizar un algoritmo de detección de errores en casos como algoritmos alojados en canales de alta confianza, ya que esto puede ayudar a revisar el contenido de nueva cuenta o para evitar una mayor pérdida en paquetes. Al igual que en servicios donde el ancho de banda es algo limitado, ayudando así algoritmos como CRC-32, que en este caso, para evitar un mayor consumo de recursos en el ordenador, al únicamente detectar el error hace más eficaz y rápida su ejecución en el código.

Conclusiones

1. El algoritmo CRC-32 es el algoritmo que obtuvo mejores resultados en las pruebas realizadas.

2. Los sockets son una excelente herramienta para poder realizar comunicación entre programas independientemente del lenguaje de programación en el que estén hechos.

3. Es importante conocer la manera y el formato en el que los algoritmos de detección y de corrección de errores serán implementados para evitar problemas de ruido o de interferencia durante la codificación y la transmisión de estos.

4. Los algoritmos de detección de errores son más amenos con los recursos del equipo, ya que no requieren de un mayor procesamiento para corregir errores como sí pueden serlo los algoritmos enfocados en la corrección de errores.

5. Se deben tomar en cuenta siempre las tasas de errores como también los bits de paridad en los algoritmos de corrección de errores, ya que si no son correctamente implementados, es probable que ocasionen varios problemas para el receptor al momento de mostrar los mensajes.

Referencias

- Python. (2024). Socket — Low-level networking interface https://docs.python.org/3/library/socket.html

- Tanenbaum, A., & Wetherall, D. (2011). Computer Networks (5th ed.). Seattle: Prentice Hall. Ch. 3.

- Windows App Development. (2 de septiembre de 2023). Getting Started with Winsock. https://learn.microsoft.com/en-us/windows/win32/winsock/getting-started-with-winsock