Universidad del Valle de Guatemala Facultad de ingeniería

Laboratorio 2 Segunda Parte: Esquemas de detección y corrección de errores

Estefania Elvira 20725
Jose Antonio Cayetano Molina 20211
Jose Pablo Monzon 20309

Guatemala, 11 de agosto del 2023

Descripción de la práctica

El objetivo principal de esa práctica es comparar el rendimiento y eficiencia de tres algoritmos dos para detección de errores (CRC-32, Fletcher) y otro para la corrección de errores (Hamming), así cómo también la simulación de un modelo de capas para la emisión y recepción de mensajes a través de un socket.

Los mensajes están compuestos por una trama, al igual que la selección del algoritmo que se utilizará para modificarlo. La trama se compone de una secuencia de bits que representan un carácter ASCII y puede incluir 32 bits adicionales, siendo el caso del algoritmo CRC-32. En el caso de hamming se generan bits representativos que se posicionan en puntos exponentes de 2, de modo que al realizar la codificación por ellos se logra que la combinación de los bits puede identificar que bit tiene una inversión, de modo que automáticamente se puede corregir al hacer el proceso inverso y realizar un “xor” o un “or” excluyente.

El emisor tiene la tarea de codificar una cantidad específica de mensajes, introducir errores en cada bit de la trama que se está codificando y luego enviar los mensajes al receptor a través de un socket. La selección del algoritmo de codificación se realiza a través de una entrada del usuario. Por su parte el receptor recibe los mensajes enviados por el emisor y utiliza el algoritmo correspondiente para detectar o corregir errores en cada trama

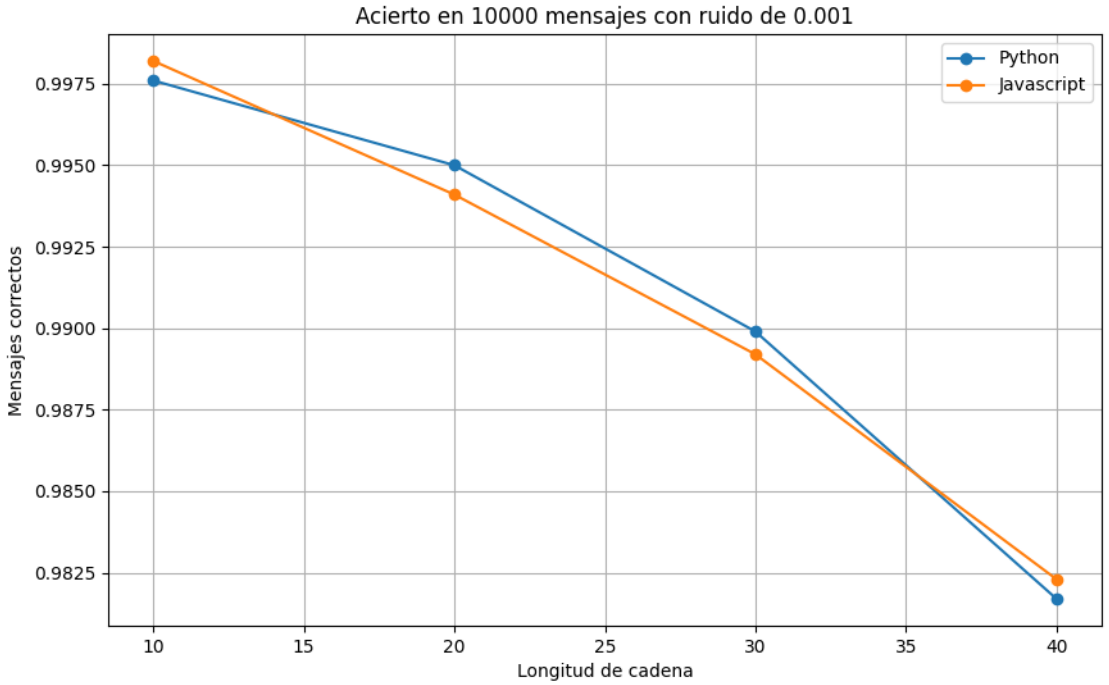
Resultados

Hamming

Para probar el algoritmo de hamming se compara el string luego de haber sido corregido con el original, y si estos no son idénticos se declara como un error.

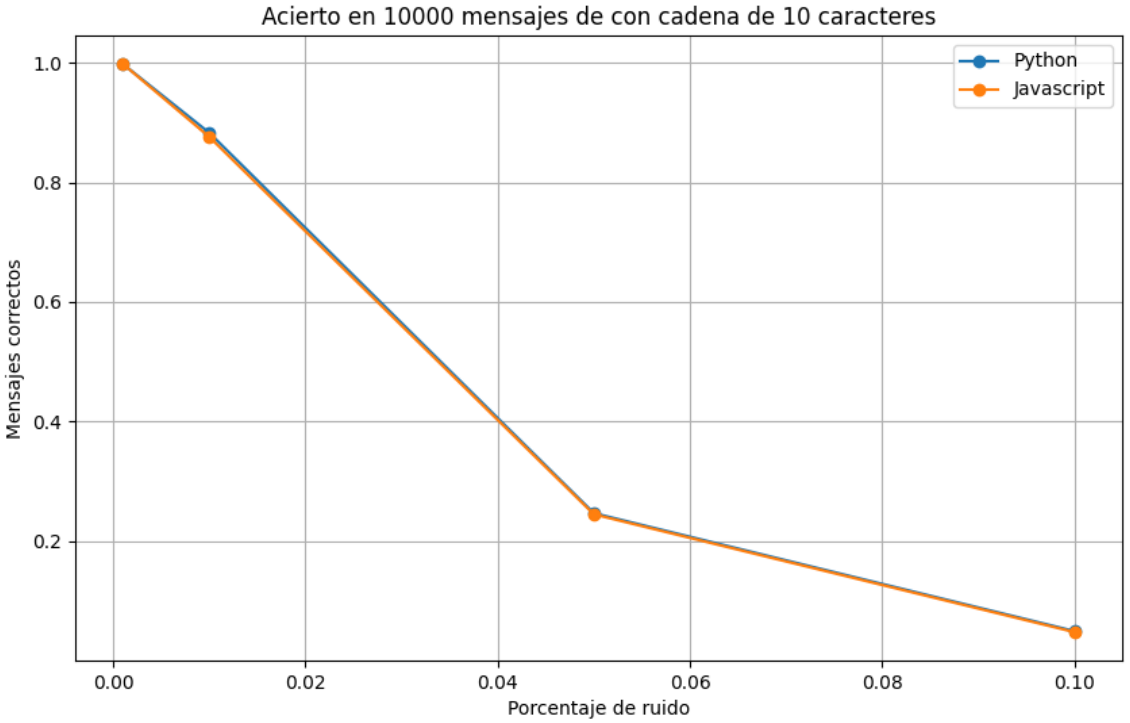
1. Prueba de 10k con ruido de 0.001 con diferentes largos de cadena

Receptor	Len = 10	Len = 20	Len = 30	Len = 40
Python	9976/10000	9950/10000	9899/10000	9817/10000
Javascript	9982/10000	9941/10000	9892/10000	9823/10000



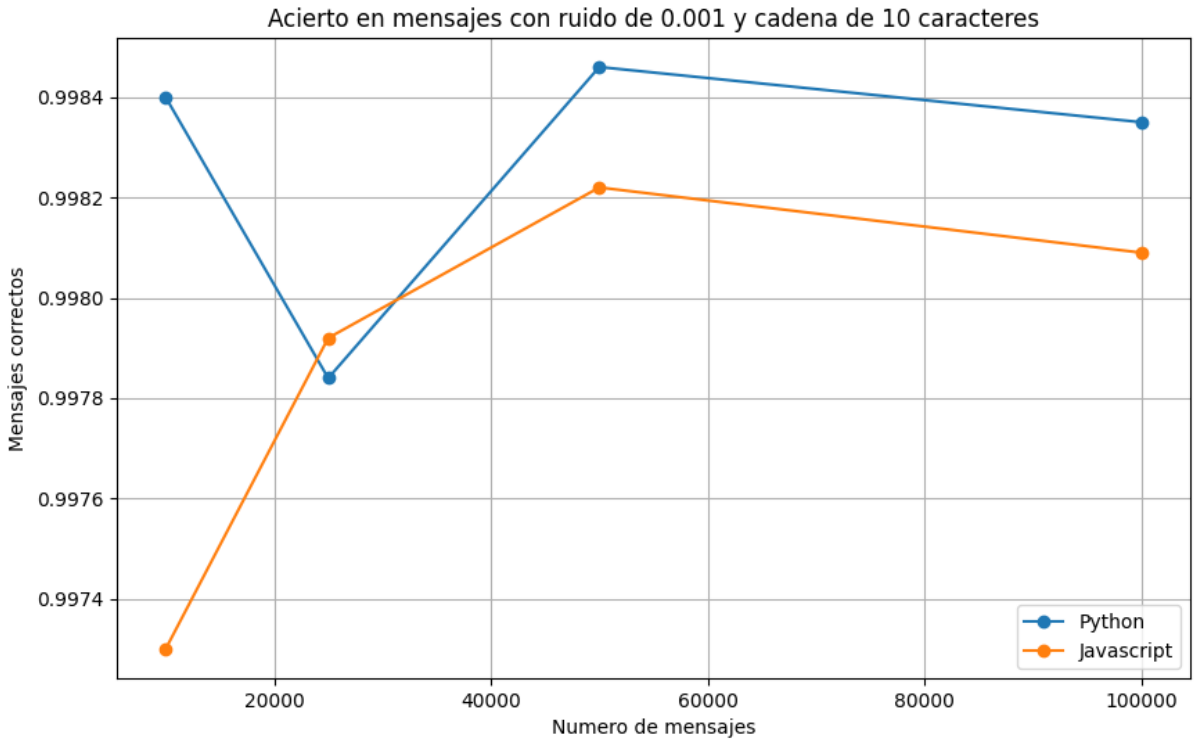
2. Prueba de 10k con cadenas de longitud 10 con diferentes ruidos

Receptor	0.001	0.01	0.05	0.1
Python	9979/10000	8827/10000	2470/10000	498/10000
Javascript	9980/10000	8759/10000	2451/10000	482/10000



3. Pruebas de cadenas de longitud 10 con ruido 0.001 con diferentes número de pruebas

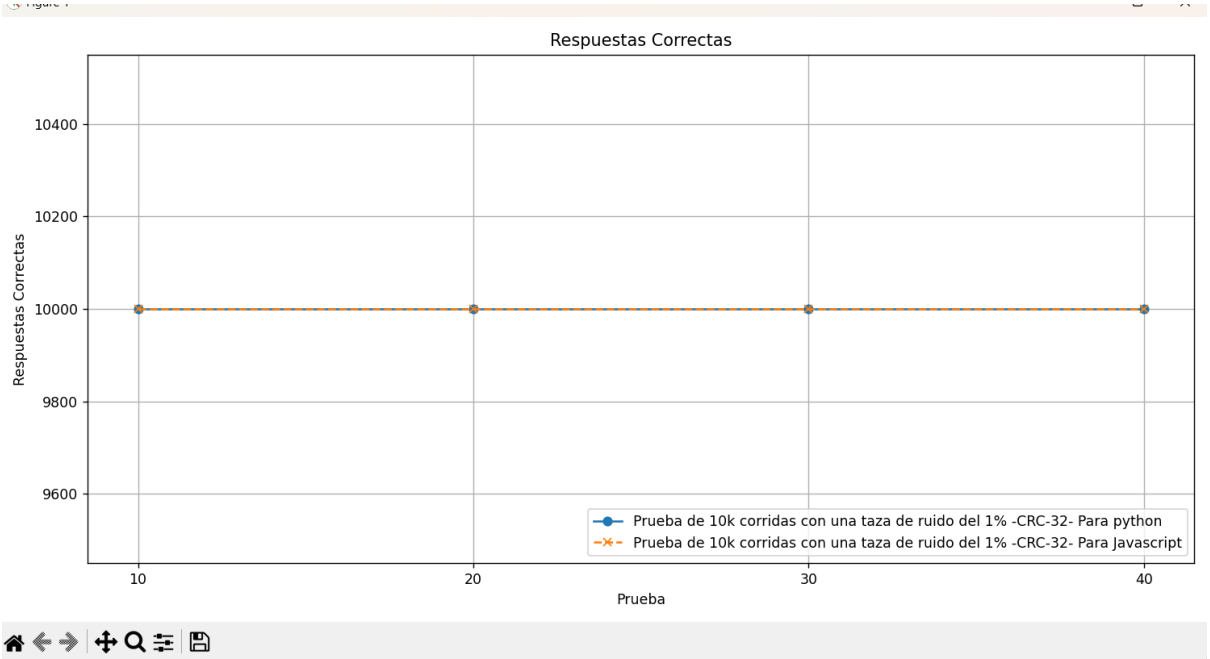
Receptor	10000	25000	50000	100000
Python	9984/10000	24946/25000	49923/50000	99835/100000
Javascript	9973/10000	24948/10000	49911/10000	99809/10000



CRC-32

1. Prueba de 10k corridas con una taza de ruido del 1%

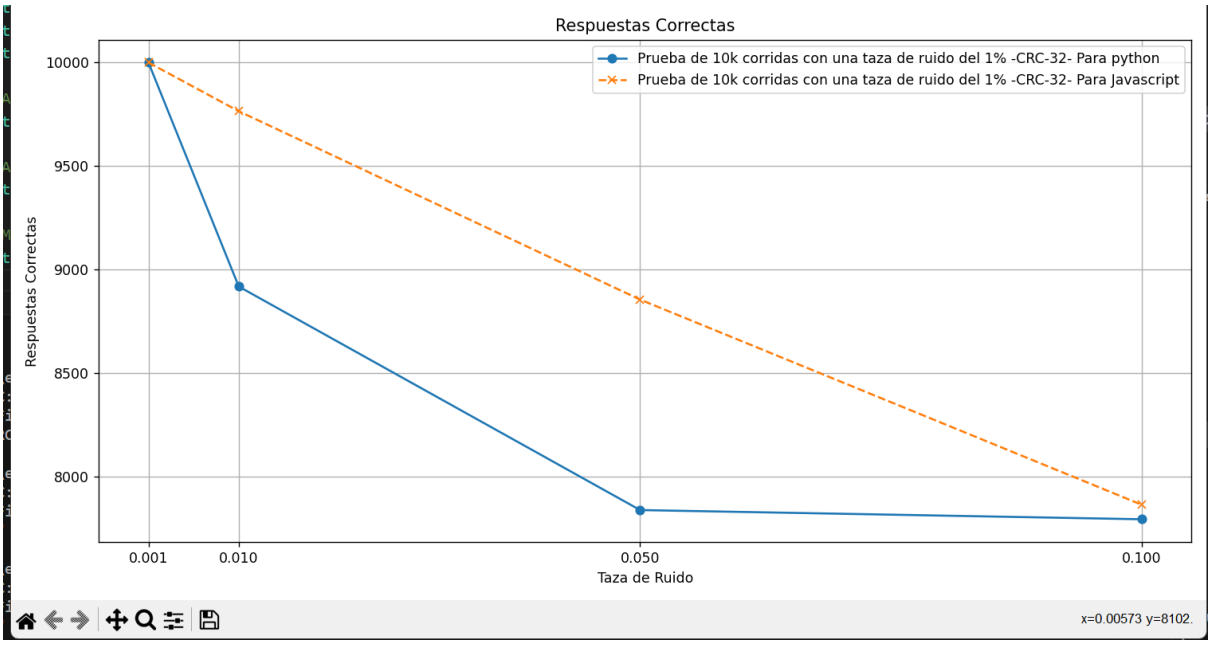
Receptor	Len = 10	Len = 20	Len = 30	Len = 40
Python	10,000.0	10,000.0	10,000.0	10,000.0
Javascript	10,000.0	10,000.0	10,000.0	10,000.0



Gráfica 1: Prueba 10k corridas con una taza de ruido del 1%, en python y javascript

2. Prueba de 10k con cadenas de longitud 10 con diferentes ruidos

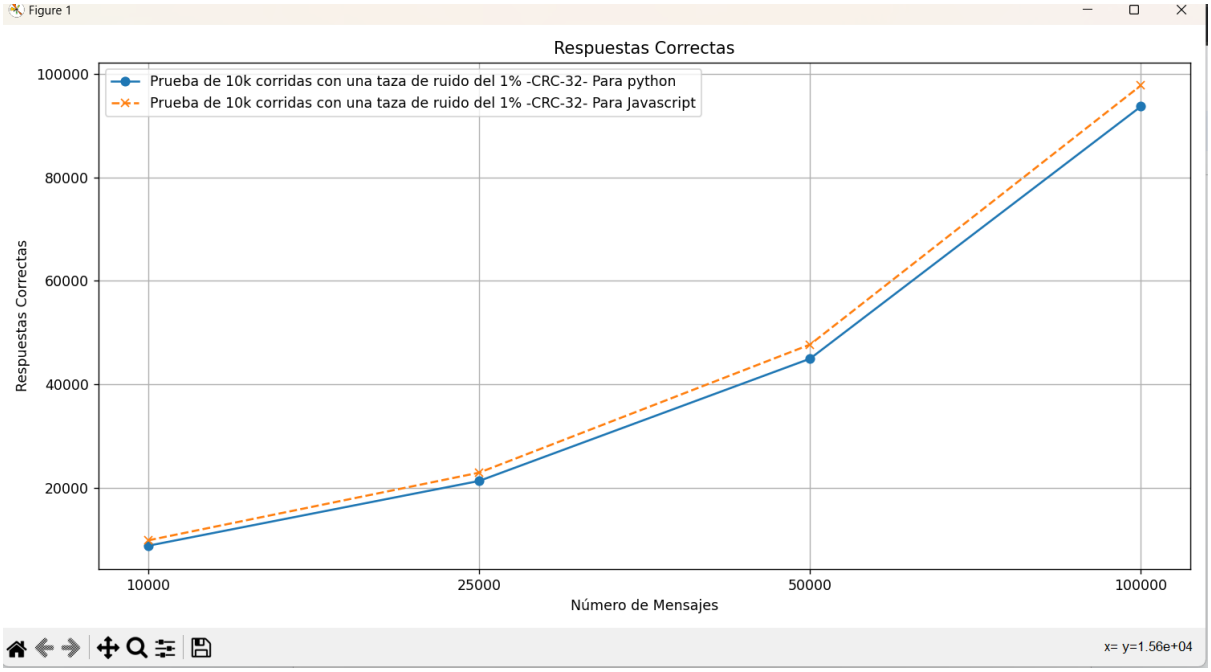
Receptor	0.001	0.01	0.05	0.1
Python	10,000.0	8,919.0	7,840.4	7,796.0
Javascript	10,000.0	9,765.0	8,856.0	7,866.0



Gráfica 2: Prueba de 10k con cadenas de longitud 10 con diferentes ruidos, en python y javascript

3. Pruebas de cadenas de longitud 10 con ruido 0.01 con diferentes número de pruebas

Receptor	10000	25000	50000	100000
Python	8,829.0	21,346.0	44,963.0	91695.0
Javascript	9,873.0	22948.0	47691.0	97759.0

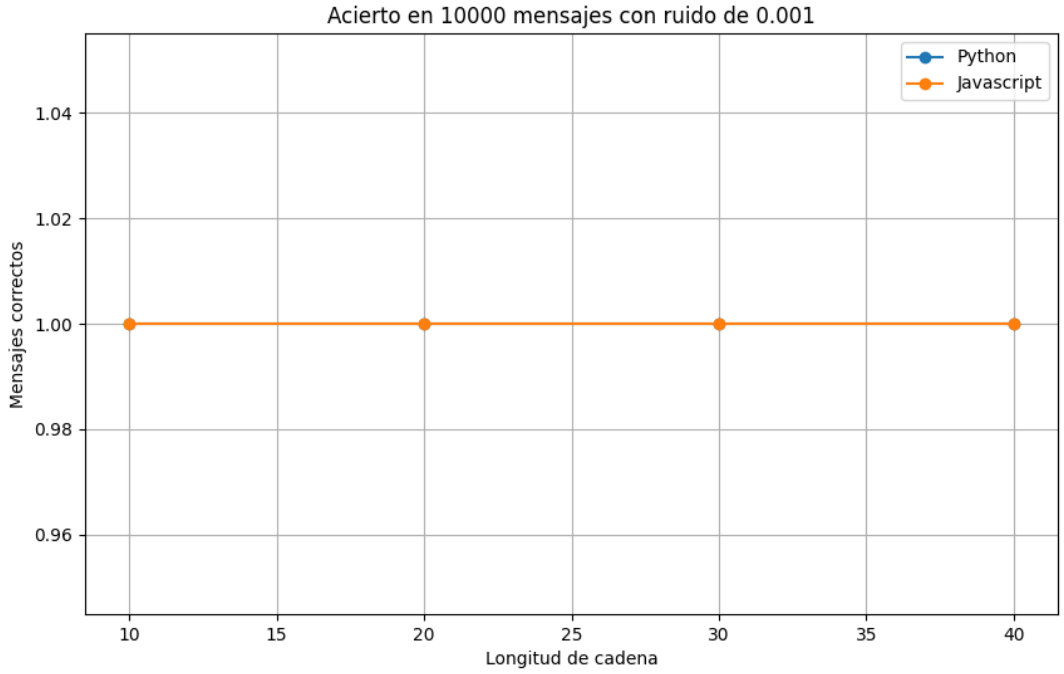


Gráfica 2: Pruebas de cadenas de longitud 10 con ruido 0.01 con diferentes número de pruebas
, en python y javascript

Fletcher Checksum:

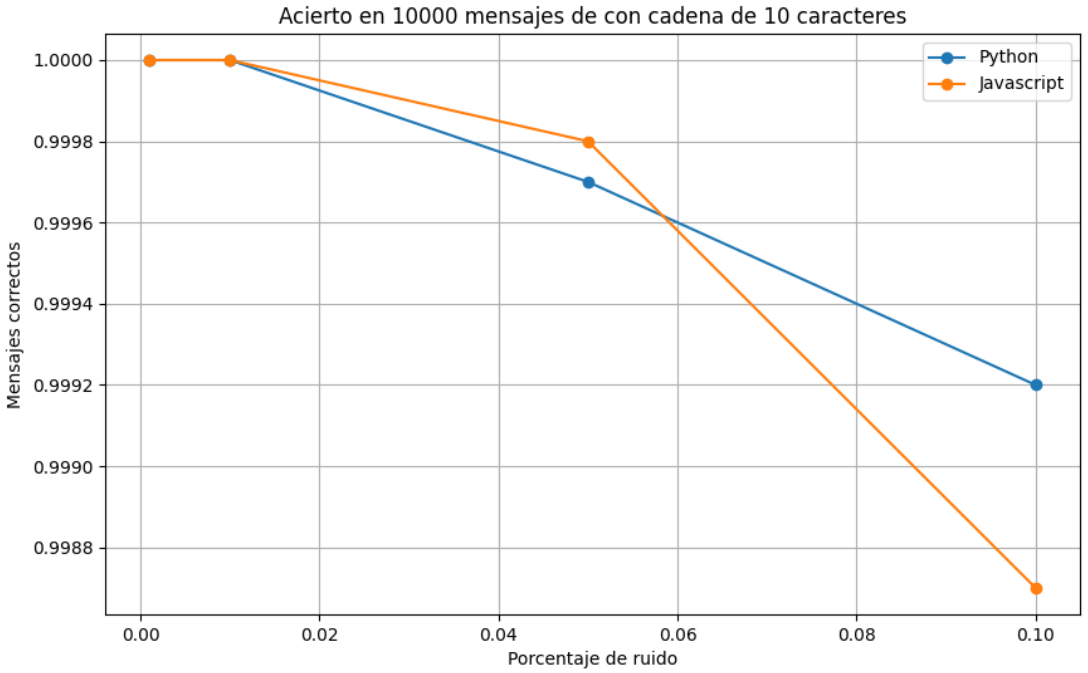
1. Prueba 10k corridas con una taza de ruido del 1%.

Receptor	Len = 10	Len = 20	Len = 30	Len = 40
Python	100%	100%	100%	100%
Javascript	100%	100%	100%	100%



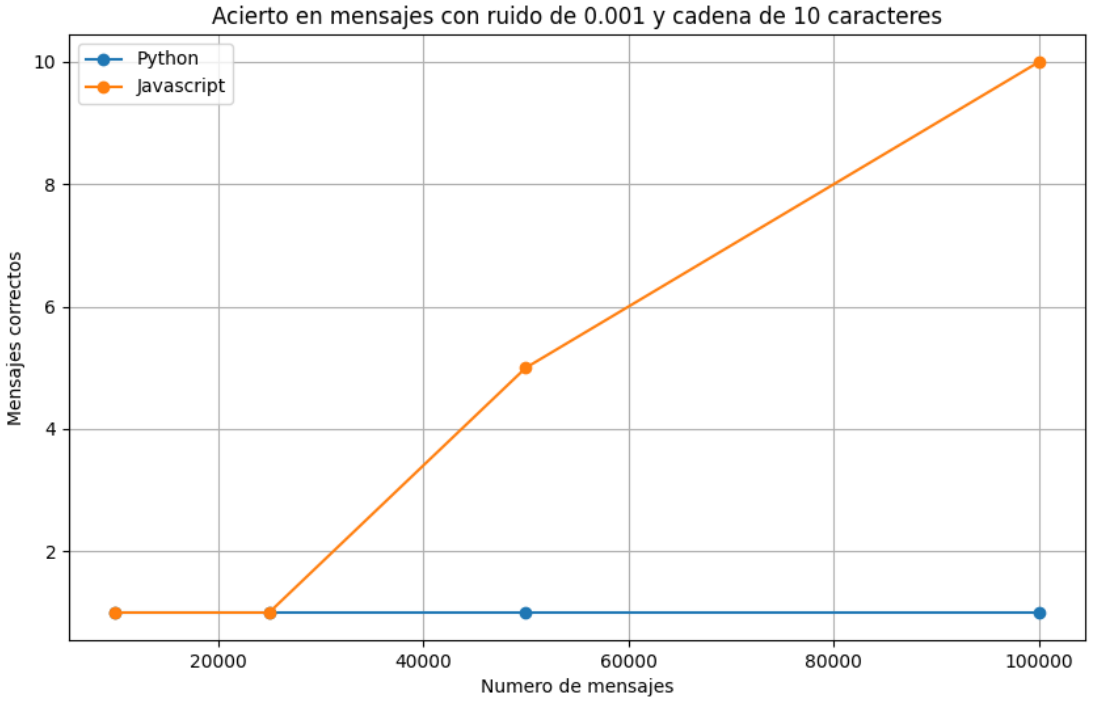
2. Prueba de 10k con cadenas de longitud 10 con diferentes ruidos

Receptor	0.001	0.01	0.05	0.1
Python	10000/10000	10000/10000	9997/10000	9992/10000
Javascript	10000/10000	10000/10000	9998/10000	9987/10000



3. Pruebas de cadenas de longitud 10 con ruido de 0.01 con diferentes número de pruebas.

Receptor	10000	25000	50000	100000
Python	10000/10000	25000/25000	49999/50000	100000/100000
Javascript	10000/10000	25000/25000	49997/50000	99988/100000



Discusión

En el mundo actual, las comunicaciones entre distintos sistemas es fundamental para que todo siga funcionando efectivamente, al momento de realizar esto hay algunos errores que pueden ocurrir al momento de este intercambio gracias al ruido e interferencias en el ambiente. De modo que pusimos a prueba tres metodologías diseñadas para enfrentar estos problemas.

Para simular el ambiente normal de comunicación entre nodos en una red se utilizó una estructura por medio de sockets para contactar múltiples lenguajes de programación cada uno conteniendo algoritmos de codificación y decodificación.

Al momento de crear los sockets se utilizó socket.io, esta herramienta nos brinda una buena opción con facilidades al momento de interconectarse desde múltiples puntos al local host, que contiene el servidor. Luego se realizaron dos clientes en cada lenguaje de programación que serían los encargados de definir el proceso de empaquetado de los mensajes, asimismo como la decodificación y agregación de ruido artificialmente.

Ahí se utiliza un patrón de factory modificado para definir el tipo de algoritmo utilizado en el mensaje, y tomando en cuenta los argumentos varios que pueden ser necesarios para la decodificación o codificación.

Para simular fielmente como son las comunicaciones en el mundo real se agregó una capa artificial de ruido que dependiendo de un porcentaje definió si un bit debería ser invertido, esto fue realizado modularmente para poder realizar pruebas.

Para las pruebas se realizaron corridas automatizadas por programa que mandaban múltiples mensajes de diferentes cadenas, con diferentes errores y se contabilizaban en el otro endpoint de modo que se pudieran hacer comparaciones entre ellos.

Se utilizaron los mismos algoritmos que en la primera parte del laboratorio, en este caso Hamming, Fletcher, y CRC. Lo que se pudo observar es que lo que más afectaba a cualquiera de estos algoritmos fue el ruido aplicado al mensaje enviado. Esto se debe a que el hecho que se cambiaran los bits hacía más difícil la tarea de tanto reconocer como cambiar los errores en caso hubiera uno.

El algoritmo que más tuvo problemas con ello fue el algoritmo de hamming en el cual en sus pruebas con un ruido de 0.1 es decir el 10% de los bits son cambiados, se obtuvo tan solo 482 respuestas correctas de las 10000. Esto se debe a que pese a que el algoritmo funciona para la corrección de errores, si existe un error que no se pueda aceptar dentro lo acordado éste será aceptado.

Conclusiones

Para el algoritmo Fletcher Checkersum lo que más le afecta en cuanto a la detección de errores es la cantidad de diferentes iteraciones, mientras más iteraciones, más mensajes se encontraran. Sin embargo, lo que más afecta al algoritmo es la cantidad de ruido que se le mete al mensaje una vez ya es enviado. Esto se debe a que mientras más bits se puedan cambiar de tal manera que el algoritmo no detecte un error, más fallos ocurrirán. No obstante, comparado con los otros algoritmos tuvo un buen rendimiento en cuanto a aciertos, esto se debe primordialmente a la suma 2 valores característicos del fletcher checksum.

El algoritmo de hamming representa al tipo de algoritmos que tienen la capacidad de corregir automáticamente los errores en los bits. Por lo que demostraron una eficiencia mayor al momento de recibir cadenas más largas. Asimismo la ventaja más grande que presenta este algoritmo sobre los otros es que al corregir la cadena de caracteres nunca genera un error en el que no se pueda mostrar la cadena, siempre se muestra el mensaje, que envés de ser totalmente descartado es mutado, de modo que en el otro punto se puede entender el mensaje correctamente.

El algoritmo de CRC-32, utilizado para la detección de errores, opera con un tiempo de ejecución constante, sin importar la cantidad de errores en un mensaje. Sin embargo, debido a su incapacidad para detectar ciertos errores en algunas tramas, los resultados de CRC-32 pueden tener un porcentaje de error más grande en comparación con otros algoritmos. Esto refleja una limitación en la precisión del algoritmo, lo cual puede ser un factor a considerar al elegir un método de detección de errores en aplicaciones críticas.

Fuentes de consulta

Socket.io. (n.d.). *Socket.IO documentation (v4.x)*. Recuperado de https://socket.io/docs/v4/

Yadav, C. (2020). Fletcher’s Checksum. https://www.tutorialspoint.com/fletcher-s-checksum

Fletcher, J. G. (1982). "An Arithmetic Checksum for Serial Transmissions". IEEE Transactions on Communications. COM-30 (1): 247–252.

Stallings, W. (2010). Comunicaciones y Redes de Computadoras. (8ª ed.). Pearson.

Tanenbaum, A. S. (2010). Computer Networks. (5ª ed.). Prentice Hall.