Universidad Del Valle de Guatemala — Departamento de Computación
Lab 2. Esquemas de detección y corrección de errores Parte 2 — Sección 20
Angel Sebastian Castellanos Pineda — 21700
Javier Alejandro Azurdia Arrecis — 21242
Diego Alejandro Morales Escobar — 21146

Descripción de la práctica

Esta práctica se centra en abordar los problemas de ruido y errores de transmisión en sistemas de comunicación. El objetivo es comprender el funcionamiento de un modelo de capas, implementar un método de comunicación para el envío de información y experimentar con la transmisión de datos incluso cuando el canal no garantiza la integridad de la información. Se pide al estudiante que, mediante el uso de varios algoritmos de corrección y detección de errores previamente desarrollados en el laboratorio anterior, implemente una simulación completa de la comunicación entre varios programas, introduciendo ruido en los mensajes enviados. Esto tiene el fin de experimentar de primera mano cómo no siempre se puede asegurar la integridad de la información enviada y así poder comparar las diferentes fortalezas y debilidades de los distintos métodos implementados.

Resultados

Para poder llevar a cabo las pruebas de implementación se utilizaron 17 mensajes, los cuales constan de los siguientes datos:

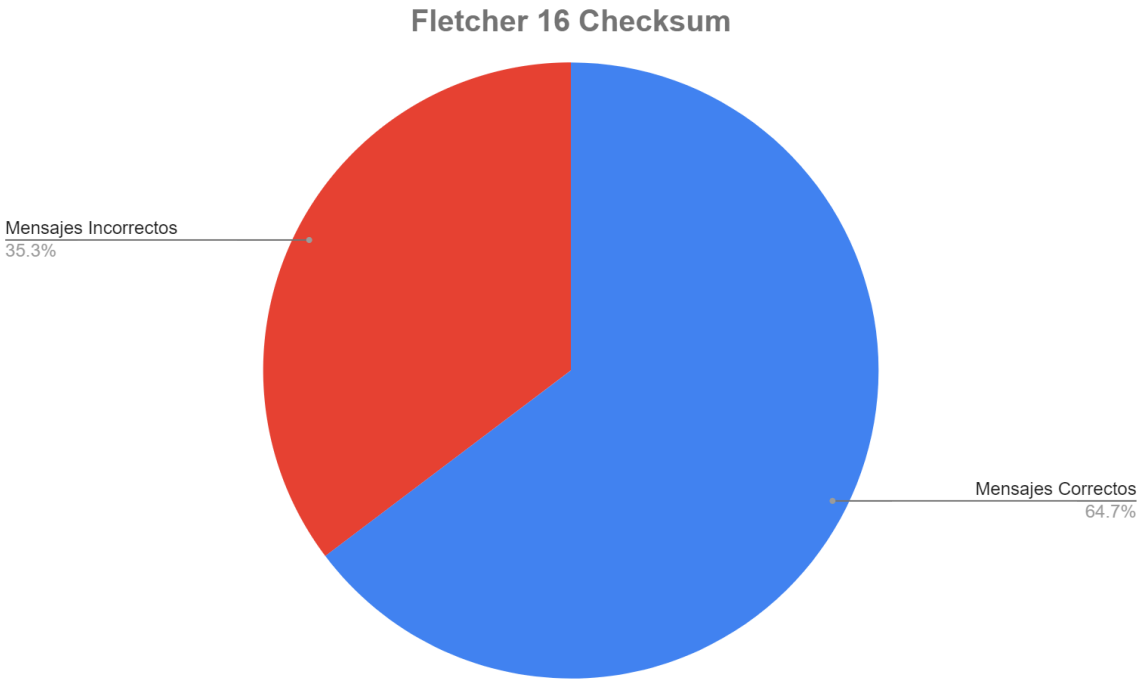
Palabra	Probabilidad de error
Hola	0.0005
Mundo	0.001
Como	0.0015
Estas	0.002
Hoy	0.0025
Es	0.003
Un	0.0025
Buen	0.002
Dia	0.0015
Para	0.001
Aprender	0.0005
Sobre	0.001

Redes	0.0015
De	0.002
Computadoras	0.0025
Y	0.003
Comunicaciones	0.008

Cabe resaltar que para la generación del cambio de bits, se utilizó una misma semilla para todas las pruebas, esto con el fin de poder reproducir el experimento.

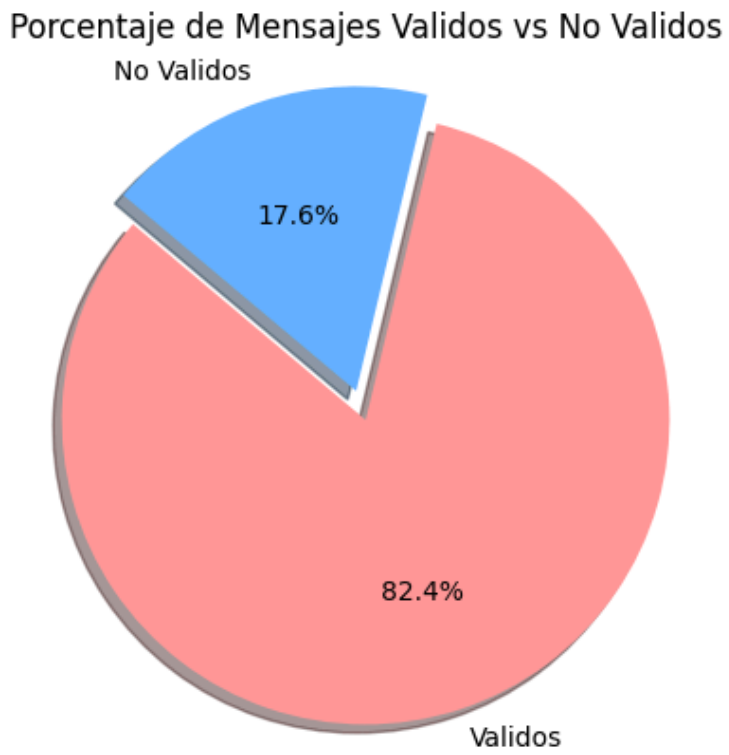
Fletcher 16 Checksum

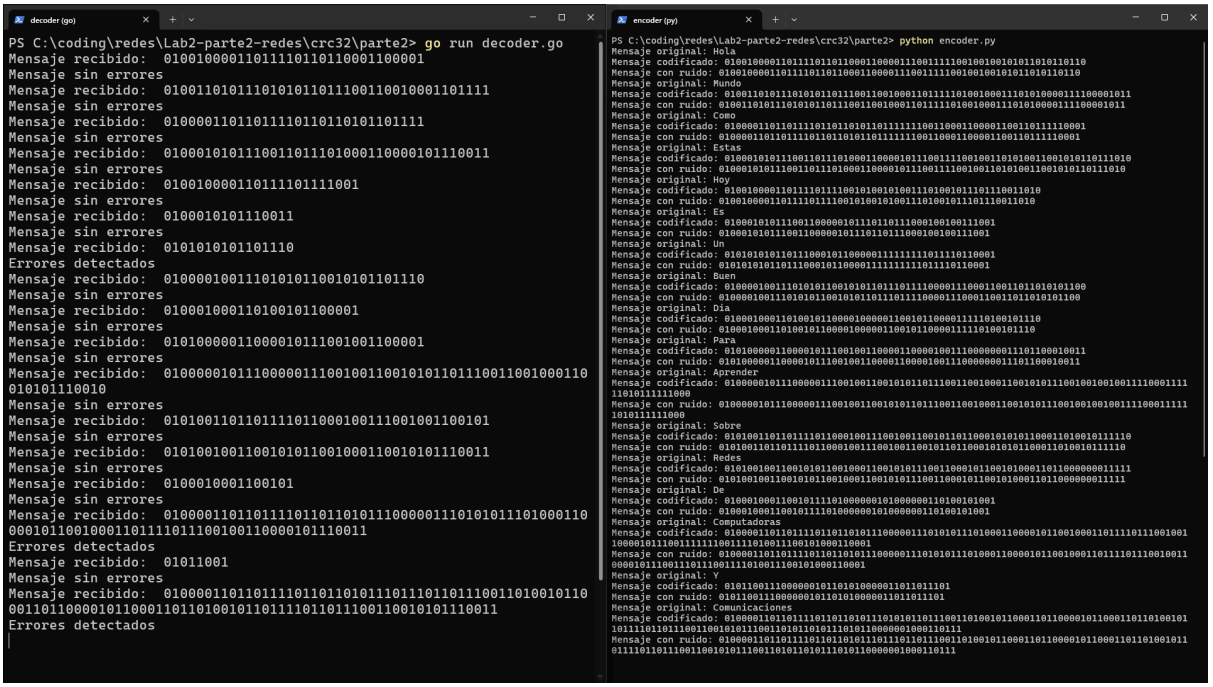
Los resultados para este algoritmo fueron los siguientes, de los 17 mensajes enviados, 11 fueron recibidos como válidos mientras que 6 fueron descartados debido a manipulación por parte de la capa de ruido. Recordando que este es un algoritmo únicamente para la detección de errores, es decir no es capaz de corregir errores, y que no se forzó la implementación para que los checksum fueran los mismos para varios mensajes, se tuvo un 64.7% de aciertos.



CRC32

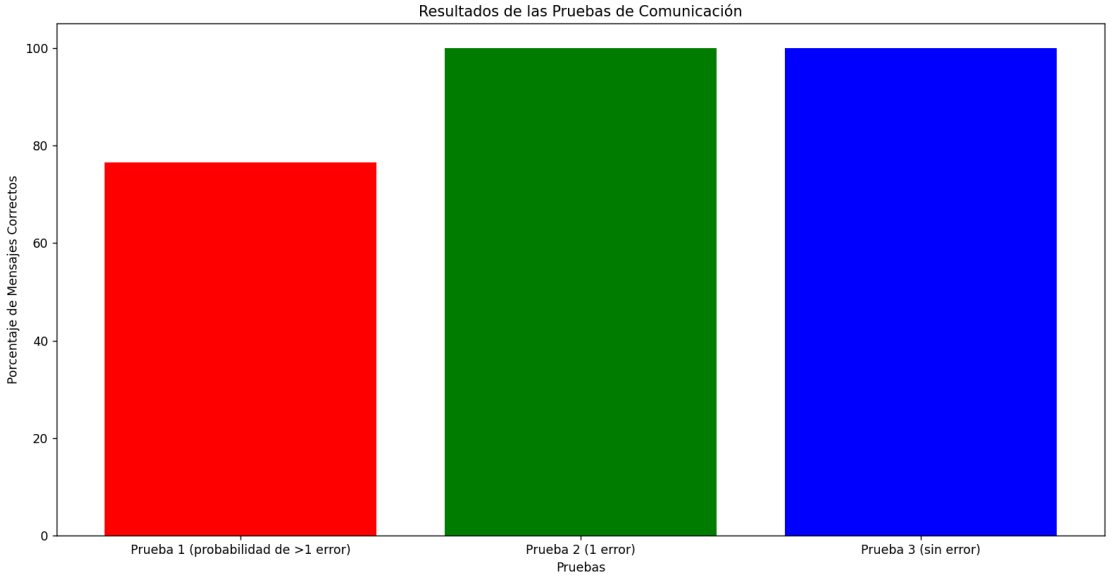
Los resultados arrojados por el algoritmo crc32, tomando en cuenta los requerimientos de ruido en la red, usando la misma semilla (0) que en las otras pruebas y tomando en cuenta que es un algoritmo de detección de errores, presentan un porcentaje de error o mensajes no válidos del 18%, frente a un porcentaje de válidos del 82%.

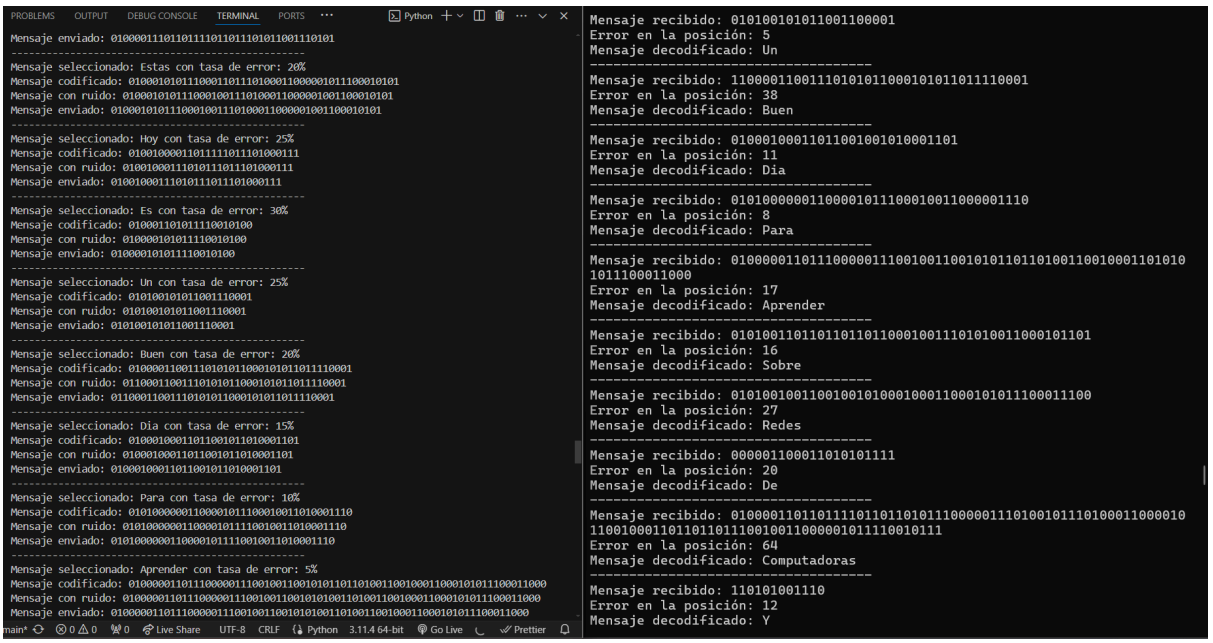




Hamming

Los resultados de las pruebas muestran que el código Hamming es altamente eficaz en la corrección de un solo error, con un porcentaje de mensajes correctos del 100% en la Prueba 2. Sin embargo, su desempeño disminuye drásticamente cuando se presentan múltiples errores en un mensaje, como se observa en la Prueba 1, donde el porcentaje de mensajes correctos es del 76%. Esto es debido a la probabilidad de que haya mensajes con más de un error. El código Hamming no puede corregir correctamente más de un error, ya que está diseñado para corregir únicamente un error por mensaje. En ausencia de errores, como se muestra en la Prueba 3, todos los mensajes se reciben correctamente.





Discusión

Los resultados del algoritmo CRC32 muestran un porcentaje de mensajes válidos del 82.4% y un porcentaje de mensajes no válidos del 17.6%. Este algoritmo es principalmente usado para detectar errores en lugar de corregirlos, lo que significa que puede identificar mensajes alterados por el ruido, pero no puede hacer nada para corregirlos. La capacidad del CRC32 para detectar mensajes corruptos de casi el 18 % de los mensajes demuestra su eficacia en la detección, asegurando que el sistema receptor no los considere válidos. Esto demuestra que CRC32 es un método efectivo para detectar errores de transmisión de datos, pero no garantiza la integridad de los mensajes recibidos si el objetivo es recibir datos sin errores.

Los resultados para el algoritmo Fletcher 16 Checksum muestran que de los 17 mensajes enviados, 11 fueron aceptados como válidos (64.7%) y 6 fueron descartados (35.3%) debido a errores. La gran cantidad de mensajes descartados demuestra que este algoritmo se enfoca en la detección de errores en lugar de en su corrección. La tasa de aciertos de Fletcher 16 es menor que la de CRC32, posiblemente debido a las diferentes metodologías de generación y verificación de la suma de comprobación. Aunque es eficaz en la detección de errores, su aplicación puede no funcionar bien en situaciones donde se necesita alta precisión y menor pérdida de datos.

Por último, los resultados de las pruebas con el código Hamming muestran una eficacia variada según la cantidad de errores presentes en los mensajes. En la Prueba 2, con una probabilidad de un solo error, el código Hamming demostró una eficacia del 100%, corrigiendo todos los errores y entregando mensajes correctos. Sin embargo, en la Prueba 1, donde existe la posibilidad de múltiples errores, la eficacia cayó al 76%, debido a que el código Hamming no puede corregir más de un error por mensaje. En la Prueba 3, sin errores, todos los mensajes fueron recibidos correctamente. Esto indica que aunque el código Hamming es excelente para la corrección de un solo error, su capacidad disminuye

significativamente cuando se presentan múltiples errores, lo que lo hace menos fiable en canales con alto nivel de ruido.

Conclusiones

La comparación de estos algoritmos muestra que cada uno tiene sus propias ventajas y limitaciones. CRC32 y Fletcher 16 Checksum son eficaces para la detección de errores, pero no ofrecen ninguna capacidad de corrección, lo cual es adecuado para aplicaciones donde la simple identificación de errores es suficiente y se pueden retransmitir los datos en caso de detección de errores. El código Hamming, por otro lado, proporciona capacidades de corrección para un solo error, sin embargo su eficacia disminuye significativamente en presencia de múltiples errores. Para la mayoría de aplicaciones y si la eficiencia del propio algoritmo lo permite, siempre es mejor usar el código Hamming.

Comentario

Este fue un laboratorio realmente interesante, ya que nos permite experimentar de primera mano lo necesarios e importantes que son este tipo de algoritmos de detección y corrección de errores para una correcta comunicación entre distintos dispositivos, y que sin ellos, prácticamente estaríamos a la merced de las interferencias, ya sean intencionales o no, y nos recalca lo compleja que puede llegar a ser algo que a simple vista pareciese algo sumamente simple para 2 personas.

Referencias

Llamas, L. (2024, abril 26). *Cómo hacer comunicaciones TCP con Node.js y el modulo NET*. Luis Llamas. https://www.luisllamas.es/comunicacion-tcp-nodejs/