Renatto Esteban Guzmán Sosa - 21646
Herber Sebastián Silva Muñoz - 21764
Redes

**Laboratorio 2 | Parte 2**

**Link al repositorio:** https://github.com/hsilv/Lab2-Redes

## Descripción de la práctica

Esta es la continuación del laboratorio anterior, en la cual pudimos enfocar los esfuerzos en vincular la comunicación del programa emisor y receptor por medio del uso de sockets. Esta implementación nos permite detectar y corregir errores al igual que experimentar cómo funcionaría la comunicación en medios expuestos a un canal no confiable. Se utilizaron los algoritmos de Hamming, un método utilizado ampliamente para detectar y corregir errores en un mensaje (JetBrains Academy, 2019) y CRC32 que utiliza una división polinómica para generar el checksum y de esa forma detectar errores (Faster Capital, 2024).

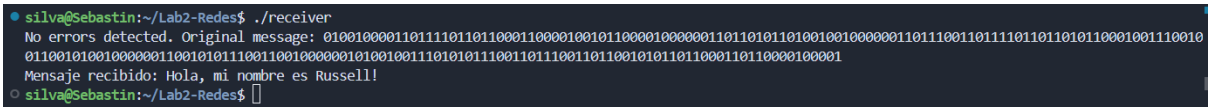
## Ejemplo de Funcionamiento

```
silva@Sebastin:~/Lab2-Redes$ python3.11 emitter.py
Ingrese un mensaje: Hola, mi nombre es Russell!
Ingrese la cantidad de errores a simular en formato (errores/bits): 1/50

[?] Seleccione el algoritmo a usar:
 > Hamming
   Cyclic Redundancy Check (CRC32)

Cadena Original:  1001100110000110011110110110001110000100101100001000000110110101110100100100000011011100110111101101101100010011100100110010110010
0000011001010111001100100000010100100111010101110011011100110110010101101100011011000100001
Cadena con ruido:  1001100110000110011110110110001110000100101100001000000110110101110100100100000011011100110111101101101100010011100100110010110011
0000001100101011100110010000001010010011101010111001101110011011001010110110001101100010000
Probabilidad de error:  0.02
Datos enviados: {"data": "1001100110000110011110110110001110000100101100001000000110110101110100100100000011011100110111101101101100010011100100110010
1011001000000110010101110011001000000101001001110101011100110111001101100101011011000100001", "algorithm": "Hamming"}
silva@Sebastin:~/Lab2-Redes$
```

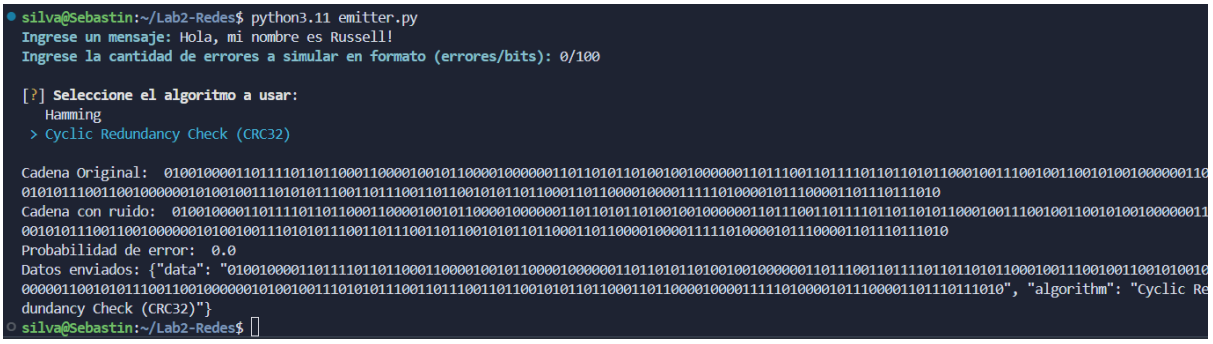
```
silva@Sebastin:~/Lab2-Redes$ ./receiver
Error detected at position: 254
Error corrected at position: 254
Multiple errors detected at positions: 2 4 8 16 32 64 128
Unable to correct errors. Message discarded.
silva@Sebastin:~/Lab2-Redes$
```

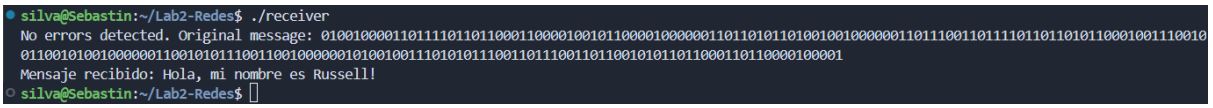
En este caso la cantidad de errores fueron varios, por lo que la integridad del mensaje ya no era para considerarse y se descartó, en este caso el algoritmo de Hamming no fue capaz de corregir tantos errores.

```
silva@Sebastin:~/Lab2-Redes$ python3.11 emitter.py
Ingrese un mensaje: Hola, mi nombre es Russell!
Ingrese la cantidad de errores a simular en formato (errores/bits): 0/100

[?] Seleccione el algoritmo a usar:
 > Hamming
   Cyclic Redundancy Check (CRC32)

Cadena Original:  1001100110000110011110110110001110000100101100001000000110110101110100100100000011011100110111101101101100010011100100110010110010
0000011001010111001100100000010100100111010101110011011100110110010101101100011011000100001
Cadena con ruido:  1001100110000110011110110110001110000100101100001000000110110101110100100100000011011100110111101101101100010011100100110010110011
0000001100101011100110010000001010010011101010111001101110011011001010110110001101100010000
Probabilidad de error:  0.0
Datos enviados: {"data": "1001100110000110011110110110001110000100101100001000000110110101110100100100000011011100110111101101101100010011100100110011
1011001000000110010101110011001000000101001001110101011100110111001101100101011011000100001", "algorithm": "Hamming"}
silva@Sebastin:~/Lab2-Redes$
```

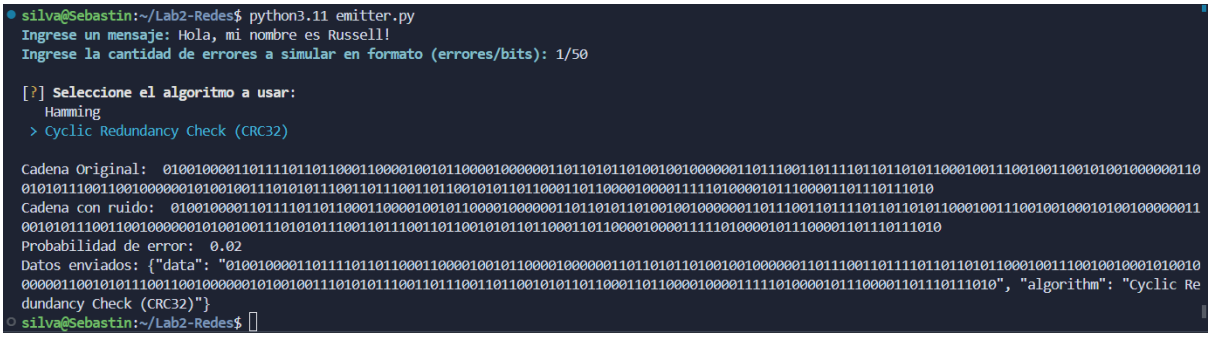


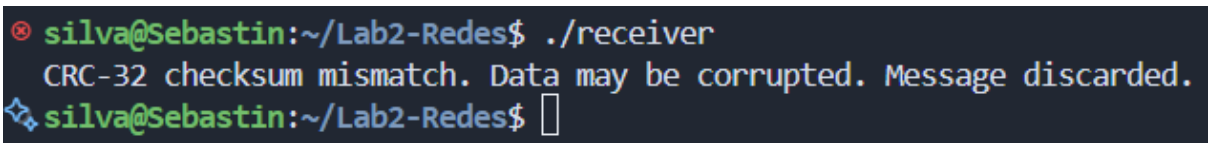

En este caso, se le dijo al emisor que enviase un mensaje con ninguna probabilidad de hacer un swap en algún bit, por lo tanto el mensaje recibido tuvo que haber sido decodificado perfectamente sin necesidad de correcciones, en este caso con el algoritmo de Hamming. Debería tener el mismo comportamiento en un algoritmo de detección como CRC32:



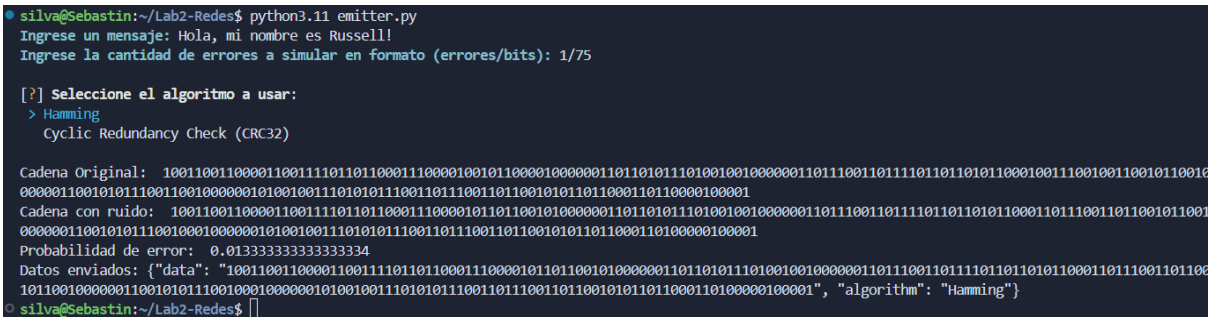




Ahora bien, cuando la cantidad de errores es grande, pero aún son corregibles el mensaje puede distorsionarse en el caso de corregirse, sin embargo, los algoritmos de detección serán capaces de rechazar el mensaje.




Ahora bien, en la corrección, ocurren distorsiones como:




Este comportamiento es natural, y es una debilidad de los algoritmos de corrección de errores.

**Resultados**

| Hamming | | | |
|---|---|---|---|
| Tamaño | Probabilidad de Error | Overhead | Falla |
| 224 | 0.02 | 8 | Sí |
| 409 | 0.02 | 9 | Sí |
| 353 | 0.02 | 9 | Sí |
| 216 | 0.005 | 8 | No |
| 361 | 0.005 | 9 | Sí |

| CRC32 | | | |
|---|---|---|---|
| Tamaño | Probabilidad de Error | Overhead | Falla |
| 248 | 0.005 | 32 | No |
| 248 | 0.02 | 32 | No |
| 409 | 0.02 | 32 | No |
| 353 | 0.02 | 32 | No |
| 216 | 0.005 | 32 | No |
| 361 | 0.005 | 32 | No |

**Decodificación sin errores:**
Tanto el algoritmo de Hamming como el de CRC-32 funcionan correctamente al momento de decodificar el mensaje cuando no se contienen errores.

Detección de errores:
Para detectar errores ambos funcionan correctamente, aunque el CRC32 tiene más capacidad de detección, ya que puede detectar todos los errores de una cadena mientras que el Hamming solo tiene la capacidad de detectar dos.

Corrección de errores:
Aquí es donde más se diferencian ambos algoritmos, ya que el único de los dos capaz de corregir los errores es el de Hamming, pero si son varios los errores la detección se puede ver afectada y mostrar un mensaje muy distorsionado.

**Discusión**

En general, ambos algoritmos funcionan y fueron diseñados para diferentes ámbitos. Además, resultan ser solo algoritmos, no adivinos, y en algunos casos es incluso natural que lleguen a fallar. En los resultados obtenidos, observamos los comportamientos esperados para el diseño de los mismos. Por ejemplo, el algoritmo de Hamming funciona muy bien cuando hay varios errores, pero su cantidad no es significativa con respecto a la longitud de la cadena, este es el comportamiento esperado, puesto que si por ejemplo, si rompemos un plato y una muñeca de porcelana, veremos porcelana en el piso, pero si hay demasiadas esquirlas, será imposible saber si la porcelana pertenecía a una muñeca o a un plato. En asuntos más técnicos, esto se traduce a que a más larga una cadena, más probabilidad hay de que pueda tener más errores, lo que reduce la efectividad de corrección del algoritmo de Hamming. Por otro lado, el algoritmo CRC32 funciona muy bien para cadenas largas como cortas, sin embargo, introduce demasiado overhead en cadenas muy cortas, por lo que en estos casos y a motivos de detección convendría más un algoritmo Hamming, sin embargo, si las cadenas son más largas, la efectividad de detección de CRC32 se mantiene intacta, por lo que sería más útil en esos casos. Entonces podría decirse, que es más conveniente usar algoritmos de detección para cadenas largas, y de corrección para cadenas cortas. Asimismo, también depende de la probabilidad de interferencia, si la probabilidad de interferencia es muy alta, o sobrepasa cierto estándar, es mejor usar algoritmos de detección, pero si es baja, conviene usar algoritmos de corrección.

**Comentario grupal sobre el tema**

Me parece interesante que no es que un algoritmo sea mejor que el otro para todo, sino que hasta cierto punto se complementan, el CRC32 es excelente para detectar errores, pero no es capaz de corregirlos, mientras que Hamming no puede detectar tantos pero igual intentará arreglar la mayor cantidad posible.

En mi opinión, este tema ilustra perfectamente cómo en tecnología (y en la vida) a menudo no hay una solución perfecta. Cada enfoque tiene sus puntos fuertes y sus limitaciones, y la elección depende del contexto y las necesidades específicas.

**Conclusiones**

- El algoritmo CRC32 es superior en la detección de errores que el algoritmo de Hamming para cadenas largas, pudiendo detectar múltiples de estos.

- El algoritmo de Hamming resalta por no solo detectar sino de corregir errores de varios bits, algo que CRC32 no puede. Sin embargo, esto está algo ligado a la naturaleza del diseño de estos algoritmos.

- El algoritmo Hamming es más eficiente en términos de Overhead que el CRC32, pero pierde robustez en cadenas más largas.

- El algoritmo CRC32 es más robusto para detectar errores en las cadenas que Hamming, esto sin importar el largo de las cadenas. Sin embargo, su overhead puede ser grande para cadenas pequeñas, debido a los 32 bits del checksum

## Referencias


- Faster Capital. (2024, June 3). *Deep Dive into CRC32: A More Robust Error Detection Mechanism*. FasterCapital. https://fastercapital.com/content/Deep-Dive-into-CRC32--A-More-Robust-Error-Detection-Mechanism.html

- JetBrains Academy. (2019, May 16). *Hamming code | Error control | String coding algorithms | Algorithms | Algorithms and structures | Computer science*. Hyperskill. https://hyperskill.org/learn/step/29178