Universidad del Valle de Guatemala
Facultad de Ingeniería
Departamento de Ciencias de la Computación
CC3067 - Redes

# Laboratorio 2 – Esquemas de Detección y Correción

Fecha de la entrega
Diego Duarte, 22075.
Juan Pablo Solis, 22102
Semestre II - 2025

## Descripción de la Practica

La práctica consistió en analizar y comparar el desempeño de los algoritmos Hamming y Fletcher-16 mediante la transmisión de 1,000 mensajes por cada longitud de caracteres en un canal con probabilidad de error, registrando la cantidad de mensajes correctos y erróneos. Luego se obtuvo la probabilidad de que el mensaje fuera erróneo según la longitud del mensaje y el algoritmo utilizado.

## Metodología Utilizada

Se definió un total de 1,000 mensajes por cada longitud de caracteres (4, 5 y 6) con el fin de estandarizar las pruebas y permitir la comparación objetiva entre ambos algoritmos. El receptor, dependiendo del algoritmo seleccionado, verificaba la integridad de cada mensaje: en el caso de Hamming intentaba corregir errores de un solo bit, mientras que con Fletcher-16 se recalculaba el checksum de 16 bits para validar la transmisión. Finalmente, se registraron los mensajes correctos y erróneos en cada escenario, calculando la probabilidad de error y analizando el impacto de la redundancia introducida por cada algoritmo sobre la eficiencia de la transmisión.

## Repositorio de GitHub

https://github.com/DiegoDuaS/DetCorSchemes

## Parte 1

## Escenarios de Prueba

### Algoritmo de Corrección - Códigos de Hamming

```
PS C:\Users\diego\OneDrive\Escritorio\2025\Semestre VIII\Redes\DetCorSchemes\Hamming> node emisor_hamming.js 1011010111
Bits de datos: 1011010111
Bits de paridad (r): 4
Bit de paridad global: 0
Código Hamming extendido: 111101100101110
```

- Sin Errores

```
estre VIII/Redes/DetCorSchemes/Hamming/receptor_hamming.py"
Ingrese el mensaje Hamming recibido: 111101100101110

Mensaje recibido (con paridad): 111101100101110
Sin errores
Mensaje original (sin paridad): 1011010111
```

- Un Error

```
estre VIII/Redes/DetCorSchemes/Hamming/receptor_hamming.py"
Ingrese el mensaje Hamming recibido: 111101110101110

Mensaje recibido (con paridad): 111101110101110
Un error en la posición 8
!!!! Corrigiendo bit en posición 8 (bit de paridad).
Mensaje corregido (con paridad): 111101100101110
Mensaje original (sin paridad): 1011010111
```

- Dos+ Errores

```
PS C:\Users\diego\OneDrive\Escritorio\2025\Semestre VIII\Redes\DetCorSchemes> &
estre VIII/Redes/DetCorSchemes/Hamming/receptor_hamming.py"
Ingrese el mensaje Hamming recibido: 110101110101110

Mensaje recibido (con paridad): 110101110101110
Detectados 2 errores o más (no corregibles)
No se puede corregir el mensaje debido a que hay 2 errores detectados.
```

- ¿Es posible manipular los bits de tal forma que el algoritmo seleccionado no sea capaz de detectar el error? ¿Por qué sí o por qué no? En caso afirmativo, demuéstrelo con su implementación

  Al principio parecía que sí se podía, porque si cambiamos 2 bits de información que afecten al mismo bit de paridad, esa paridad no cambiaría y parecería que no hay error. Pero con 3 bits de datos y 2 o 3 bits de paridad, siempre hay otro bit de paridad que también se ve afectado. Eso obliga a cambiar más bits para que todo cuadre, y en ese punto ya no es el mismo mensaje, sino otro distinto válido.

- En base a las pruebas que realizó, ¿qué ventajas y desventajas posee cada algoritmo con respecto a los otros dos? Tome en cuenta complejidad, velocidad, redundancia (overhead), etc.

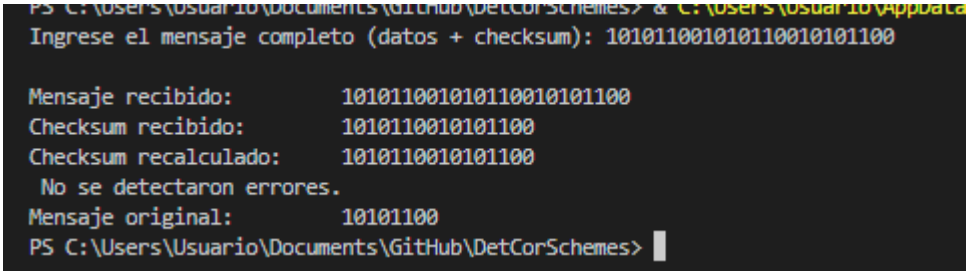
  El código de Hamming es muy simple y rápido de usar, porque solo necesita hacer sumas mod 2 (o XOR) de ciertos bits para detectar y corregir errores. Su ventaja principal es que puede corregir automáticamente un error en el mensaje y, además, agrega relativamente pocos bits de paridad, por lo que el overhead es bajo en mensajes cortos.

  La desventaja es que solo puede corregir un error; si hay dos o más errores, no siempre los detecta y no puede corregirlos. También, a medida que el mensaje se hace más largo, se necesitan más bits de paridad para mantener la detección, lo que aumenta el overhead. Por eso es más adecuado para canales con pocos errores.

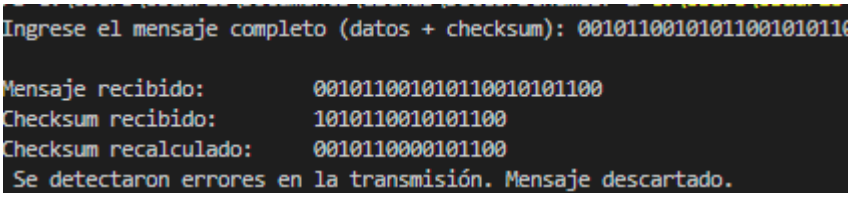
**Algoritmo de Detección**

```
PS C:\Users\Usuario\Documents\GitHub\DetCorSchemes> cd .\Fletcher_Crc\
PS C:\Users\Usuario\Documents\GitHub\DetCorSchemes\Fletcher_Crc> node emisor_fletcher_legacy.js 10101100
Mensaje original:         10101100
Checksum (Fletcher-16):   1010110010101100
Mensaje final transmitido: 1010110010101100101010100
```

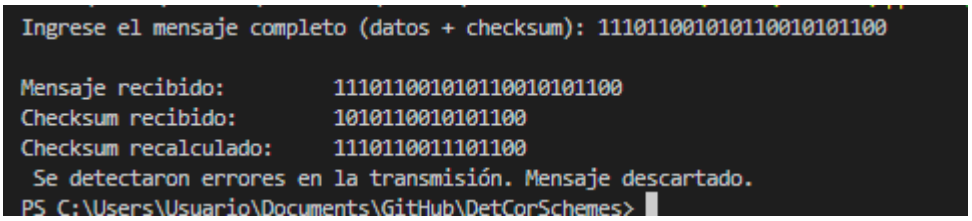
- Sin Errores




- Un Error




- Dos+ Errores




- ¿Es posible manipular los bits de tal forma que el algoritmo seleccionado no sea capaz de detectar el error? ¿Por qué sí o por qué no? En caso afirmativo, demuéstrelo con su implementación
  Si, es posible manipular el algoritmo para que no sea capaz de detectar el error. Esto se puede realizar si se alteran dos o más bytes de forma compensada, las sumas parciales (sum1 y sum2) pueden quedar iguales y el checksum no cambia. En esos casos, el receptor no detecta el error, aunque los datos estén alterados

- En base a las pruebas que realizó, ¿qué ventajas y desventajas posee cada algoritmo con respecto a los otros dos? Tome en cuenta complejidad, velocidad, redundancia (overhead), etc.
- El algoritmo Fletcher-16 es sencillo de implementar y muy rápido, ya que únicamente requiere sumas acumulativas módulo 255 sobre los bytes del mensaje. Su ventaja principal es que ofrece buena detección de errores simples con un costo computacional bajo, además de que agrega un overhead pequeño de 16 bits sin importar la longitud del mensaje.

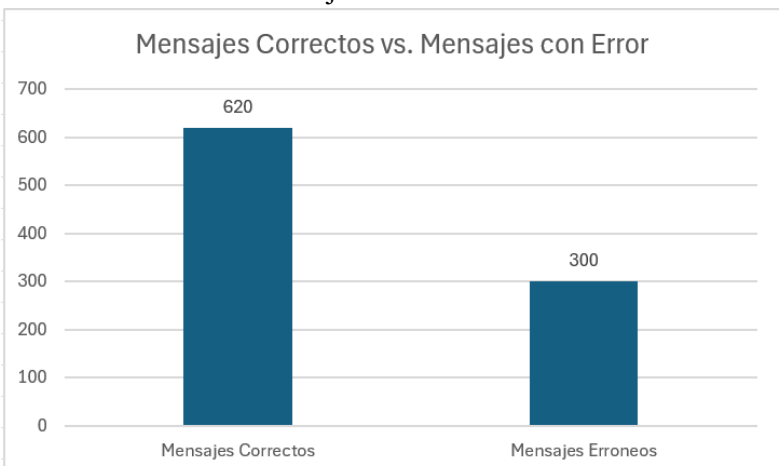
La desventaja es que no garantiza detectar todos los errores: existen combinaciones de cambios en los datos que pueden mantener las sumas parciales iguales, haciendo que algunos errores pasen inadvertidos. Por ello, aunque es eficiente en velocidad y espacio, su confiabilidad es limitada frente a errores más complejos o manipulaciones intencionales.

## Parte 2

## Resultados

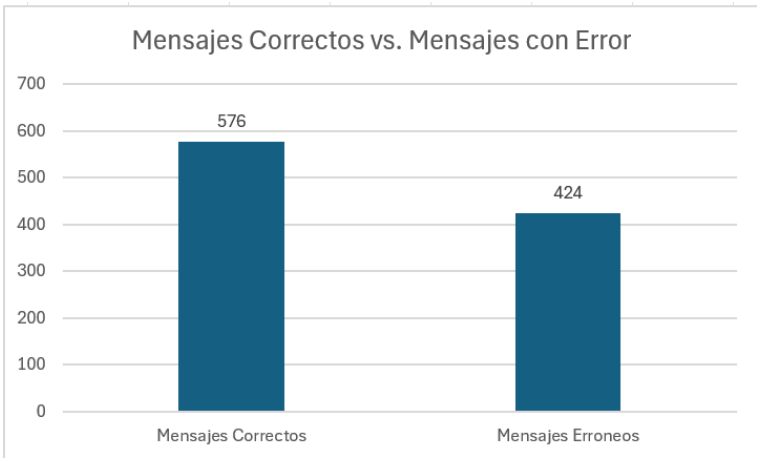
Algoritmo Fletching Checksum:

Mensaje de 4 caracteres



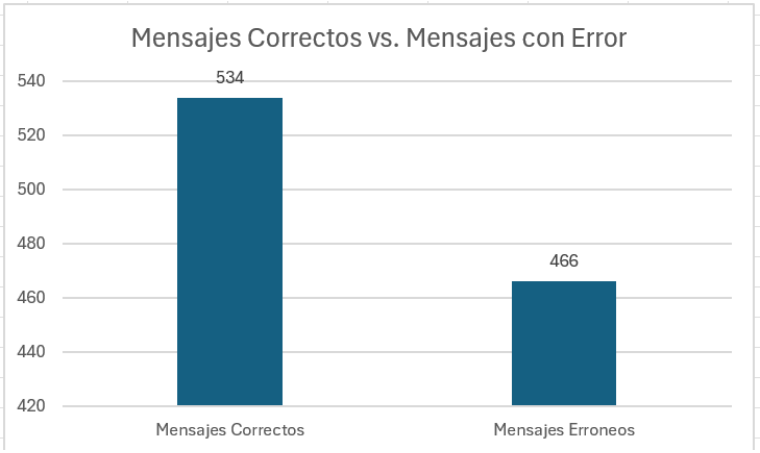

Probabilidad de Error: $\frac{300}{920} = 32.6\%$

Mensaje de 5 caracteres



Probabilidad de Error: $\frac{424}{1000} = 42.4\%$

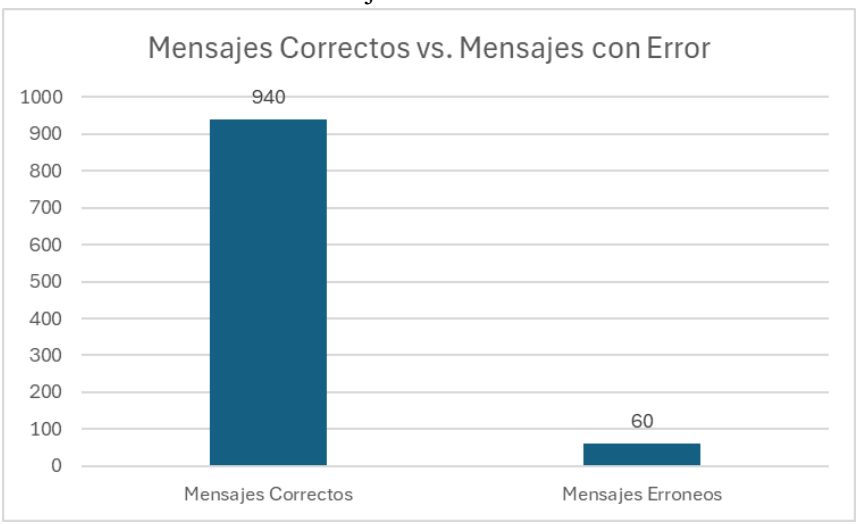
Mensaje de 6 caracteres




Probabilidad de Error: $\frac{466}{1000} = 46.6\%$

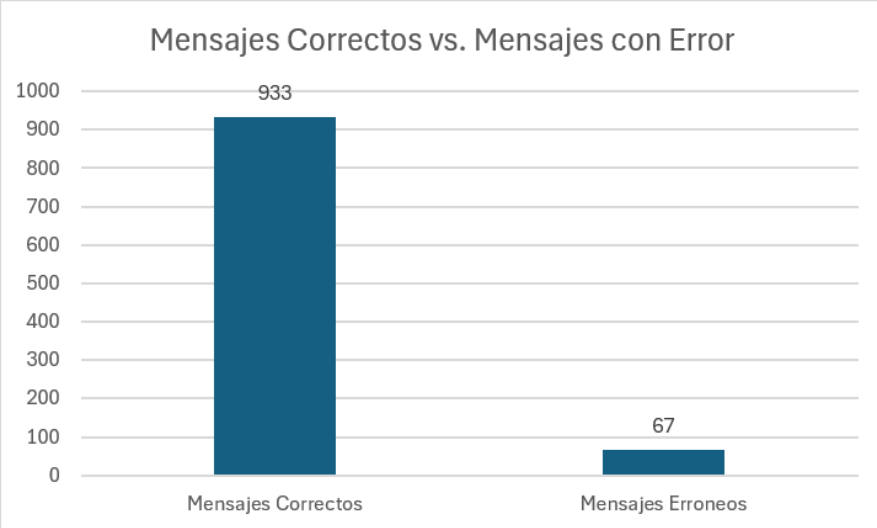
Algoritmo de Haming

Mensaje de 4 caracteres



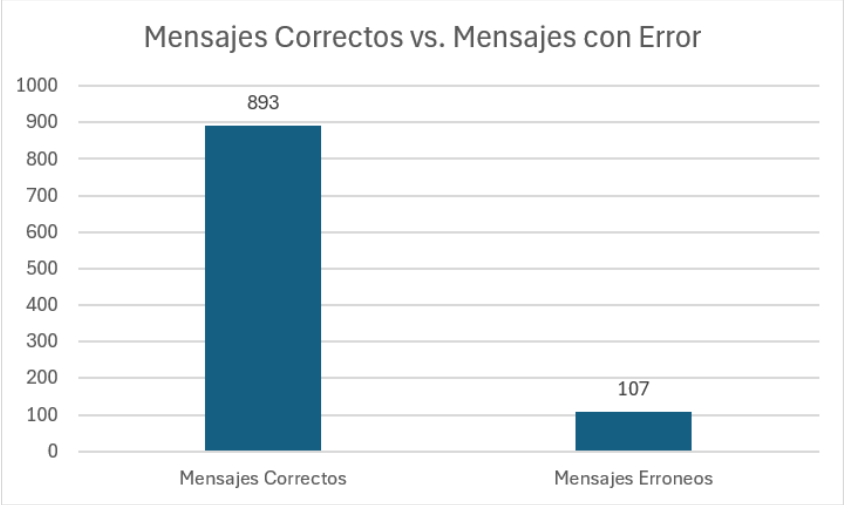

Probabilidad de Error: $\frac{60}{1000} = 6\%$

Mensaje de 5 caracteres




Probabilidad de Error: $\frac{67}{1000} = 6.7\%$

Mensaje de 6 caracteres




Probabilidad de Error: $\frac{107}{1000} = 10.7\%$

## Discusión

Para las pruebas de los distintos algoritmos se utilizaron 1,000 mensajes por cada longitud de carácter, con el objetivo de estandarizar al máximo los resultados obtenidos.

### Fletcher-16

Con mensajes de 4 caracteres, se recibieron 620 transmisiones correctas y 300 erróneas. Aquí se aprecia que la mayoría de los mensajes logra llegar de forma íntegra, lo que refleja la eficacia del algoritmo para la detección de errores en mensajes cortos.

Cuando la longitud aumentó a 5 caracteres, los mensajes erróneos ascendieron a 424, mientras que los correctos fueron 576. Aunque se observa un incremento en los errores, la diferencia con respecto al caso de 4 caracteres no es desproporcionada, y la mayoría de las transmisiones aún resultó válida.

Finalmente, en los mensajes de 6 caracteres, los errores se situaron en 466 y los correctos en 534. El número de mensajes correctos sigue siendo significativo, lo que indica que el incremento de errores no fue excesivo.

En general, se evidencia una tendencia creciente en la cantidad de errores conforme aumenta la longitud del mensaje. Esto se debe a que, al incrementarse el número de caracteres, también crece la probabilidad de que alguno de sus bits individuales sea alterado durante la transmisión.

**Algoritmo de Hamming**

En el caso de los mensajes de 4 caracteres, se recibieron 940 correctos frente a solo 60 erróneos. Esto refleja un rendimiento bastante alto, ya que el algoritmo fue capaz de detectar y corregir errores simples en la gran mayoría de las transmisiones.

Con los mensajes de 5 caracteres, se registraron 933 correctos y 67 erróneos. Aunque el número de errores aumentó ligeramente en comparación con el caso anterior, la diferencia es mínima y confirma que Hamming mantiene una alta efectividad para detectar y corregir errores en mensajes cortos.

En los mensajes de 6 caracteres, los errores aumentaron a 107 frente a 893 correctos. Aquí se aprecia con mayor claridad que conforme se extiende la longitud del mensaje, también crece la probabilidad de que ocurran errores múltiples (por ejemplo, dos bits volteados dentro de la misma palabra). Dado que Hamming solo puede corregir un error y detectar algunos casos de dos errores, cuando se producen múltiples alteraciones simultáneas su capacidad de corrección se ve superada.

No obstante, la ventaja de Hamming radica precisamente en su función de corrección de un solo bit. Incluso en un canal con ruido, muchos de los mensajes que habrían sido descartados bajo un algoritmo de solo detección, como Fletcher-16, fueron corregidos con éxito. Esto explica por qué, aun con la tendencia a que el número de errores aumente al crecer el tamaño del mensaje, la proporción de mensajes correctos es consistentemente más alta en Hamming que en Fletcher. Estos resultados se refuerzan con la probabilidad de que un mensaje sea erróneo, usando la misma longitud de caracteres en cada algoritmo, en donde el valor más alto del algoritmo de hamming es un 10.7% de que el mensaje sea erróneo, mientas que con el de Fletcher es un 46.6%.

En cuanto a redundancia por parte de los algoritmos, en el caso de Hamming, el algoritmo inserta varios bits de paridad distribuidos en posiciones específicas, más un bit de paridad global, lo que hace que la redundancia crezca conforme aumenta la longitud del mensaje, pero a cambio permite detectar y corregir errores de un solo bit; por ejemplo, en un código (7,4) se usan 3 bits de paridad

para 4 bits de información, con una tasa de código de 0.57. En contraste, Fletcher-16 genera un checksum fijo de 16 bits independientemente de la longitud del mensaje, lo que implica que su redundancia es constante y más eficiente en mensajes largos, aunque solo ofrece detección de errores y no corrección. En consecuencia, Hamming sacrifica eficiencia de transmisión para ganar capacidad de corrección, mientras que Fletcher-16 prioriza una baja tasa de redundancia estable, pero sin corrección, por lo que la elección entre ambos depende de si se requiere corrección automática (Hamming) o solo detección ligera y eficiente (Fletcher-16).

## Conclusión

Se puede concluir que los resultados muestran que el algoritmo de Hamming supera de manera consistente a Fletcher-16 en términos de confiabilidad, ya que no solo detecta errores, sino que también corrige aquellos de un solo bit, reduciendo significativamente la proporción de mensajes erróneos. Aunque su desventaja radica en el aumento de redundancia y la imposibilidad de corregir múltiples errores simultáneos, sigue siendo más adecuado en entornos con alta probabilidad de ruido. Por otro lado, Fletcher-16 destaca por su simplicidad y eficiencia en términos de redundancia, especialmente en mensajes largos, pero al limitarse únicamente a la detección de errores su desempeño resulta menos robusto en condiciones adversas, haciendo que su uso sea más conveniente cuando se prioriza la ligereza y se tolera retransmisión en caso de fallos.

## Comentarios

- La implementación de Hamming y Fletcher-16 fue directa gracias a la lógica clara de cada algoritmo; Hamming permite corregir un error simple y Fletcher detecta errores mediante checksum.
- Al aplicar ruido simulado, se observó que incluso un pequeño porcentaje de bits alterados puede corromper totalmente un mensaje si no se cuenta con corrección de errores, evidenciando la fragilidad de la transmisión sin protección.