

UNIVERSIDAD DEL VALLE DE GUATEMALA

Facultad de Ingeniería

Redes – Jorge Yass



Laboratorio 2. Parte 2

Francis Aguilar – 22243

Angela García – 22869

Introducción

En cualquier sistema de comunicación digital, los errores de transmisión representan uno de los principales desafíos a enfrentar. Estos errores pueden surgir por múltiples factores como interferencias, ruido o fallos en el medio de transmisión, lo que compromete la integridad de los datos enviados. Por esta razón, a lo largo del tiempo y con la evolución del Internet, se han desarrollado diversos algoritmos y mecanismos diseñados específicamente para detectar y corregir errores durante la transmisión de datos.

Descripción de la práctica

Esta práctica de laboratorio tiene como objetivo el comprender de una manera profunda el funcionamiento de un modelo de capas y sus servicios, implementando sockets para la transmisión de información y lograr experimentar la transmisión de información expuesta en un canal no confiable usando los esquemas de detección y corrección de errores anteriormente usados en la práctica previa. Que son programas que simulan tanto el emisor como el receptor, y utilizando distintos lenguajes de programación para cada uno.

Resultados

Algoritmo de detección: crc32

Mensaje correcto:

```
PS C:\Users\Francis\OneDrive - UVG\Francis\2025\Semestre 8\Redes\lab2-p2\deteccion> python cliente.py
Ingrese mensaje para enviar: Holaaa
Ingrese cantidad de bits para agregar ruido, 0 si no: 0
Mensaje enviado: 01001000011011110110011000010110000101100001001101110001110100101110111010
p2/deteccion_server.py - PS C:\Users\Francis\2025\Semestre 8\Redes\lab2-p2\deteccion>
Esperando mensaje...

Mensaje recibido: 01001000011011110110011000010110000101100001001101110001110100101110111010
Verificación CRC32: Mensaje correcto.

Mensaje decodificado: Holaaa
```

Mensaje con error:

```
PS C:\Users\Francis\OneDrive - UVG\Francis\2025\Semestre 8\Redes\lab2-p2\deteccion> python cliente.py
Ingrese mensaje para enviar: Holaa
Ingrese cantidad de bits para agregar ruido, 0 si no: 1
Mensaje enviado: 010010000110111111011000110000101100001110011100111101000111010101000
PS C:\Users\Francis\OneDrive - UVG\Francis\2025\Semestre 8\Redes\lab2-p2\deteccion>

Mensaje recibido: 010010000110111111011000110000101100001110011100111101000111010101000
Verificación CRC32: Error detectado en la transmisión.
```

Algoritmo de corrección: Hamming

Mensaje correcto:


```
Cliente
Ingrese el mensaje que desea mandar (escriba '1' para salir): hola
01101000011011110110110001100001
Bits de paridad necesarios: 6
Total de bits en código Hamming: 38
Posicion 1: Paridad (se calculara)
Posicion 2: Paridad (se calculara)
Posicion 3: Dato 0 (bit 0 de datos)
Posicion 4: Paridad (se calculara)
Posicion 5: Dato 1 (bit 1 de datos)
```

```
Agregar ruido? (s/n): n
Transmision sin errores
Mensaje enviado.
```

```
S8 (P8): -> 8 unos (par) -> S8=0
S16 (P16): -> 10 unos (par) -> S16=0
S32 (P32): -> 2 unos (par) -> S32=0
Sindrome: 0
No hay errores detectados
Bits de datos decodificados: 01101000011011110110110001100001
Texto reconstruido: 'hola'
Mensaje sin errores
```

Mensaje con error:

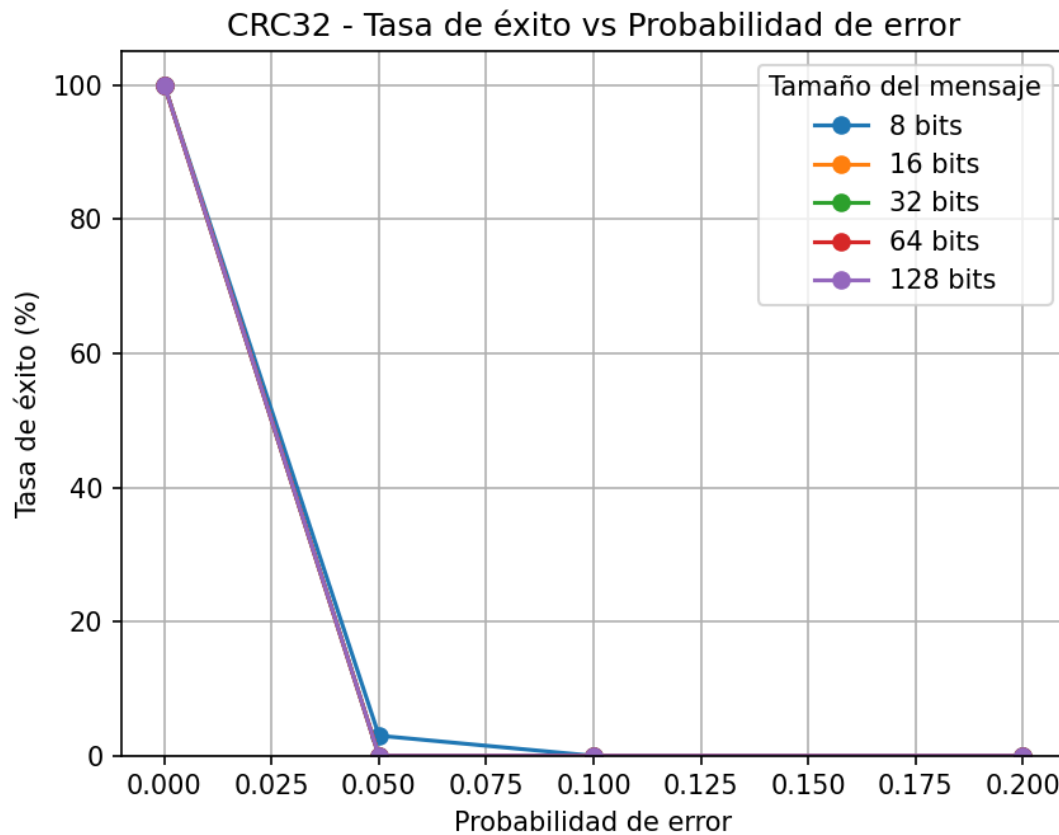
```
Agregar ruido? (s/n): s
Cuantos errores introducir? (1-3): 2
Mensaje original: 1101110000010110001000110100101010111101110011
Ingrese posicion del error 1 (1-based):
```

it Graph [faquilarleal \(41 minutes ago\)](#)  Ln 19

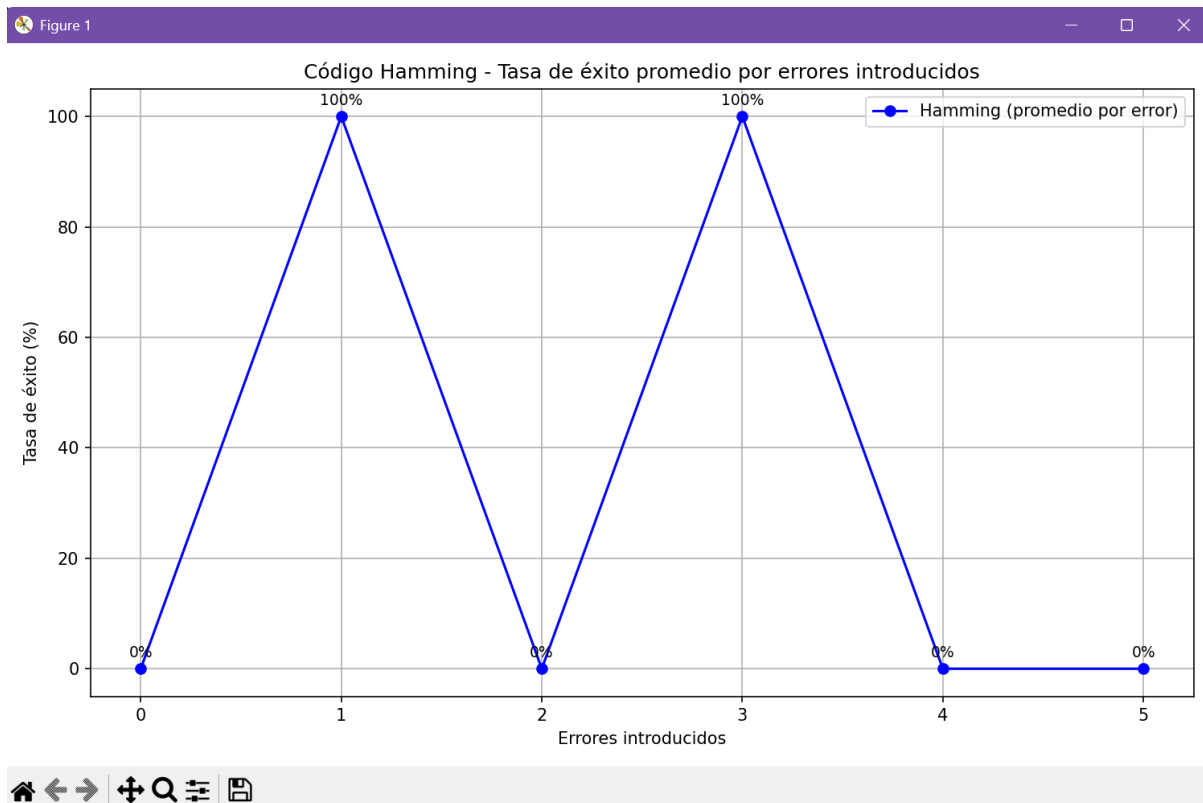
```
El codigo Hamming solo puede:
1. Corregir 1 error de forma confiable
2. Detectar (pero NO corregir) 2 errores
3. Con 3+ errores: comportamiento impredecible y usualmente lo suele empeorar
Bits de datos decodificados: 0110000101100100011010010110111101110011
Texto reconstruido: 'adios'
Se corrigió 1 error en la posición: 1
```

Gráficos

Figure 1



CRC32 es muy efectivo para detectar errores aislados en entornos con baja probabilidad de error, pero su capacidad disminuye notablemente conforme aumentan los errores, independientemente del tamaño del mensaje. Por ello, en entornos más ruidosos o críticos, podría requerirse un algoritmo de detección más robusto o técnicas de corrección adicionales.



Por otra parte, el código Hamming la tasa de éxito es del 100% cuando hay 1 error porque este es capaz de corregir un único bit, pero cuando hay más de un error la tasa cae a 0%, excepto cuando algunos casos donde el tercer error coincide con la misma posición, entonces se anula efectivamente. Y en el caso de 0, la tasa es de 0% porque no se introdujo errores.

Respuesta a las preguntas mencionadas

- ¿Qué algoritmo tuvo un mejor funcionamiento?
 - El algoritmo de Hamming tuvo un mejor funcionamiento en entornos donde es necesario corregir errores automáticamente, ya que puede detectar y corregir errores de un solo bit. En cambio, CRC es más eficiente en detectar errores múltiples, pero no los corrige. Por tanto, el mejor funcionamiento depende del criterio evaluado: si se valora la corrección automática, Hamming es superior; si se busca detectar una mayor variedad de errores con alta eficiencia y bajo costo computacional, CRC es más adecuado.
- ¿Qué algoritmo es más flexible para aceptar mayores tasas de errores?
 - El algoritmo de Hamming tuvo un mejor funcionamiento en entornos donde es necesario corregir errores automáticamente, ya que puede detectar y corregir errores de un solo bit. En cambio, CRC es más eficiente en detectar errores múltiples, pero no los corrige. Por tanto, el mejor funcionamiento depende del criterio evaluado: si se valora la corrección automática, Hamming es superior; si se busca detectar una mayor variedad de errores con alta eficiencia y bajo costo computacional, CRC es más adecuado.

- ¿Cuándo es mejor utilizar un algoritmo de detección errores en lugar de uno de corrección de errores?
 - o Es mejor utilizar un algoritmo de detección de errores como CRC cuando el sistema de comunicación permite la retransmisión de datos de manera rápida y económica. Esto es común en redes modernas donde un simple aviso de error permite reenviar el mensaje. Por el contrario, algoritmos de corrección como Hamming son preferibles en sistemas donde la retransmisión no es posible o es costosa, como en la transmisión de datos espaciales o almacenamiento físico.

Discusión

Los sistemas de comunicación tienen una arquitectura de capas, estos sirven para organizar y estructurar el proceso de transmisión de datos de una manera modular, escalable y eficiente. Cada capa tiene una función específica que permite que se pueda trabajar en una capa sin afectar a las demás o actualizar una capa sin tener que rehacer el programa. Además, una aplicación no necesita saber cómo es que se envían los bits, el usuario solo se encarga de enviar y recibir mensajes. Las capas que se tienen son las siguientes: aplicación, presentación, enlace, ruido y transmisión. Cada una de estas tiene servicios que se implementaron en los algoritmos por parte del emisor y receptor.

Entonces, se realizaron pruebas para evaluar el comportamiento de cada uno de los algoritmos. Empezando con el algoritmo de CRC32, el cual opera en la capa de enlace, se encargó de añadir redundancia al mensaje mediante un código de verificación calculado a partir del contenido del mensaje original. Este valor CRC permite que el receptor verifique si hubo errores durante la transmisión. Se observó que CRC32 es muy eficiente para detectar errores múltiples en bloques grandes de datos, pero no tiene capacidad de corrección, por lo que cualquier error detectado requiere una retransmisión del mensaje.

Por otro lado, el algoritmo de Hamming, también implementado en la capa de enlace, permitió tanto la detección como la corrección de errores de un solo bit. Esto lo hace especialmente útil en entornos donde los errores son frecuentes y la retransmisión no es viable o es costosa. Sin embargo, su rendimiento se limita a errores simples, ya que no puede corregir errores múltiples ni detectar todos los errores de dos bits.

Ambos algoritmos fueron probados frente a distintos escenarios de ruido, simulando la capa física donde se pueden alterar los bits del mensaje. Los resultados mostraron que mientras CRC32 es muy preciso al identificar errores, Hamming destaca al corregirlos cuando son simples. Finalmente, la capa de presentación se encargó de codificar y decodificar los datos (por ejemplo, traduciendo el mensaje binario a caracteres), y la capa de aplicación actuó como interfaz con el usuario final.

Comentario grupal sobre el tema

Es interesante conocer un poco de todo lo que llevan los sistemas de comunicación y cómo es que existen varios algoritmos para la detección y corrección de errores. Y cómo es que las capas son muy importantes para la modularidad, abstracción, reutilización, estandarización y

escalabilidad. Considerando todos los servicios que conllevan, tanto de parte del emisor y receptor.

Las capas son fundamentales para lograr modularidad, abstracción, reutilización, estandarización y escalabilidad. Además, facilitan el diseño, la implementación y el mantenimiento de los sistemas de comunicación, permitiendo que cada parte se pueda actualizar o mejorar sin afectar el resto. Considerando todos los servicios que conllevan, tanto del lado del emisor como del receptor, es evidente la importancia de su correcta implementación para asegurar una transmisión de datos confiable.

Conclusiones

- Se comprendió el funcionamiento de modelos de capas y sus servicios, implementando emisor y receptor con cada una de sus capas correspondientes.
- Se experimentó la transmisión de información por medio de un canal no confiable y se usó sockets para enviar y recibir la trama de información por medio del puerto elegido.
- CRC es más eficiente en detectar errores múltiples, pero no los corrige.

Citas y referencias

Hamming, R. W. (1950). Error detecting and error correcting codes. *Bell System Technical Journal*, 29(2), 147–160. <https://doi.org/10.1002/j.1538-7305.1950.tb00463.x>

GeeksforGeeks. (2021, July 28). *Hamming Code - Error detection and error correction*. <https://www.geeksforgeeks.org/hamming-code-error-detection-and-error-correction/>

Williams, R. (1993). A painless guide to CRC error detection algorithms. Retrieved from https://zlib.net/crc_v3.txt