

UNIVERSIDAD DEL VALLE DE GUATEMALA

CC3067- Redes

Sección 21

Ing. Jorge Andrés Yass Coy



Esquemas de detección y corrección de errores Parte 2

Juan Angel Carrera, 20593

Jose Mariano Reyes, 20074

GUATEMALA, 10 de Agosto de 2023

Descripción de la práctica y metodología utilizada

La práctica consiste en desarrollar una aplicación por capas para el envío y recepción de mensajes a través de sockets, con capacidad para simular ruido y errores durante la transmisión. Las capas incluyen:

- Capa de aplicación: Solicita el texto del mensaje y el algoritmo de integridad al remitente. Muestra el mensaje recibido o el error al receptor.
- Capa de presentación: Codifica el texto en binario ASCII en el lado del emisor. Decodifica el binario ASCII en texto en el receptor, comprobando si hay errores.
- Capa de enlace: Calcula la información de integridad mediante el algoritmo elegido y la concatena con el mensaje. Verifica la integridad en el receptor para detectar errores. Integra los algoritmos de detección y corrección de errores de la primera parte del laboratorio.
- Capa de ruido: Simula el ruido cambiando aleatoriamente los bits del mensaje con una probabilidad determinada.
- Capa de transmisión: Envía y recibe mensajes a través de sockets utilizando el puerto elegido.

La metodología consiste en

- Implementar la arquitectura de capas en una aplicación emisora y receptora.
- Realizar múltiples pruebas (10k, 100k+) variando el tamaño del mensaje, la probabilidad de error, el algoritmo utilizado y los parámetros del algoritmo.
- Recopilación automática de estadísticas y generación de gráficos para analizar los resultados.
- Comparación de algoritmos en términos de funcionalidad y flexibilidad para gestionar errores.
- Debatir cuándo es mejor la detección y cuándo la corrección.
- Presentación de informes sobre la aplicación, los resultados, el debate y las conclusiones.

Resultados obtenidos

CRC

```
mensaje = solicitar_mensaje()
mensaje_codificado = codificar_mensaje(mensaje)
integridad = calcular_integridad(mensaje_codificado)
trama = mensaje_codificado + integridad
trama_con_ruido = aplicar_ruido(trama, 0.01)
enviar_informacion(trama_con_ruido)
```

En la función de aplicar_ruido() estamos aplicando el ruido, si no lo queremos solo lo cambiamos por la trama normal.

Pruebas normales **sin** ruido en la trama

```
PS C:\Users\Jose\Documents\GitHub\Fork\Lab1_Redes> python .\parte2\crc\emisor.py
Ingrese el mensaje a enviar: mensaje
PS C:\Users\Jose\Documents\GitHub\Fork\Lab1_Redes> █
```

```
○ Mensaje recibido y verificado: mensaje
█
```

Pruebas normales **con** ruido en la trama

```
PS C:\Users\Jose\Documents\GitHub\Fork\Lab1_Redes> python .\parte2\crc\emisor.py
Ingrese el mensaje a enviar: mensaje
PS C:\Users\Jose\Documents\GitHub\Fork\Lab1_Redes> █
```

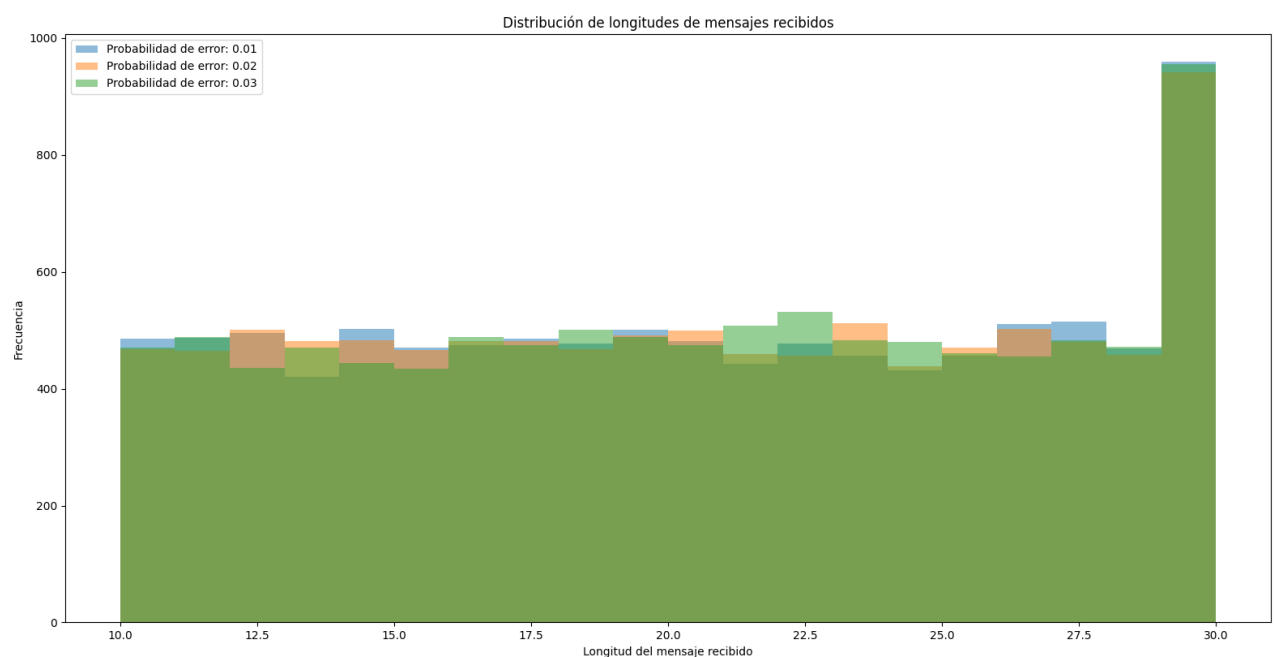
```
Error de integridad/ruido detectado en la trama recibida.
█
```

Resultados carios con ruido 0.01

```
Ingrese el mensaje a enviar: hola
PS C:\Users\Jose\Documents\GitHub\Fork\Lab1_Redes> python .\parte2\crc\emisor.py
Ingrese el mensaje a enviar: h
PS C:\Users\Jose\Documents\GitHub\Fork\Lab1_Redes> python .\parte2\crc\emisor.py
Ingrese el mensaje a enviar: capaz
PS C:\Users\Jose\Documents\GitHub\Fork\Lab1_Redes> python .\parte2\crc\emisor.py
Ingrese el mensaje a enviar: si}
PS C:\Users\Jose\Documents\GitHub\Fork\Lab1_Redes> python .\parte2\crc\emisor.py
Ingrese el mensaje a enviar: trama
PS C:\Users\Jose\Documents\GitHub\Fork\Lab1_Redes> python .\parte2\crc\emisor.py
Ingrese el mensaje a enviar: errorrrrrr
PS C:\Users\Jose\Documents\GitHub\Fork\Lab1_Redes> █
```

```
Error de integridad/ruido detectado en la trama recibida.
Mensaje recibido y verificado: h
Mensaje recibido y verificado: capaz
Mensaje recibido y verificado: si}
Mensaje recibido y verificado: trama
Error de integridad/ruido detectado en la trama recibida.
█
```

Pruebas con 10k de datos



HAMMING

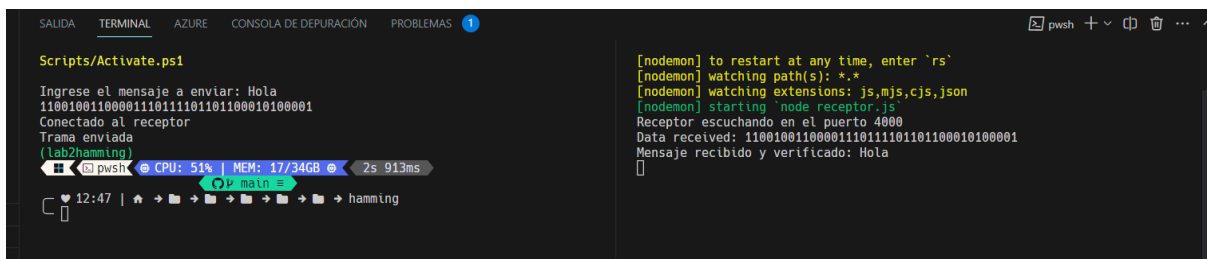
```
mensaje = solicitar_mensaje()
mensaje_codificado = string_to_bits(mensaje)
trama = hamming_codificar(mensaje_codificado)
print(trama)
trama_con_ruido = aplicar_ruido(trama)
sio = socketio.Client()

@sio.on('connect')
def on_connect():
    print('Conectado al receptor')
    sio.emit('end', trama)
    print('Trama enviada')
    sio.disconnect()

sio.connect('http://localhost:4000')
```

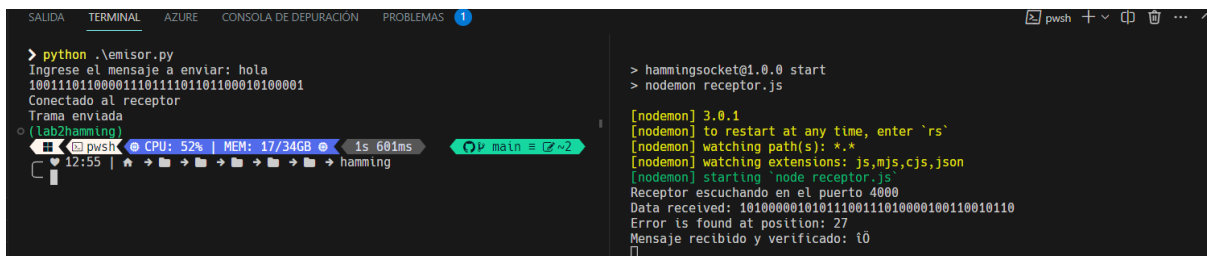
Use la misma función para generar el ruido que en crc

Pruebas normales sin ruido en la trama:



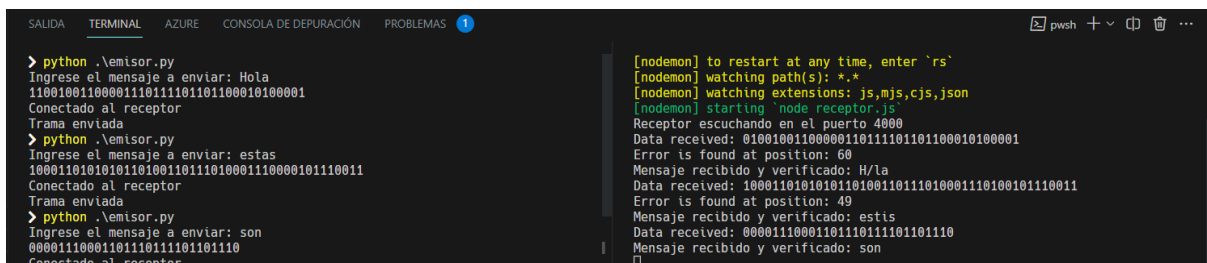
The terminal window shows the execution of a script named 'Scripts/Activate.ps1'. The user enters the message 'Hola' and the system outputs the binary message '11001001100001110111101101100010100001'. The message is then encoded into a Hamming code '11001101100001110111101101100010100001'. The encoded message is sent to the receiver, and the receiver outputs the same message, confirming successful transmission without noise.

Pruebas normales **con** más de un error en la trama:



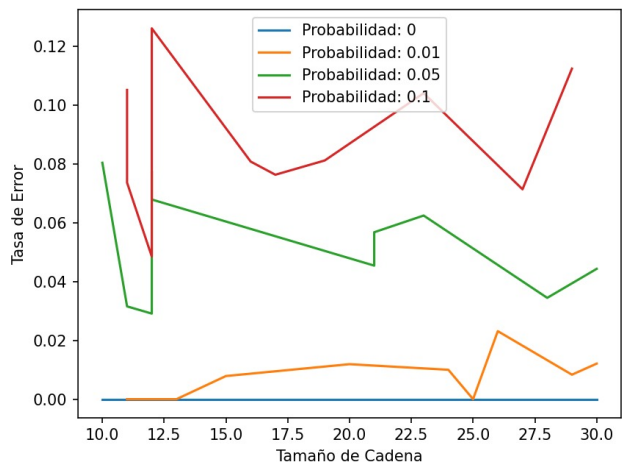
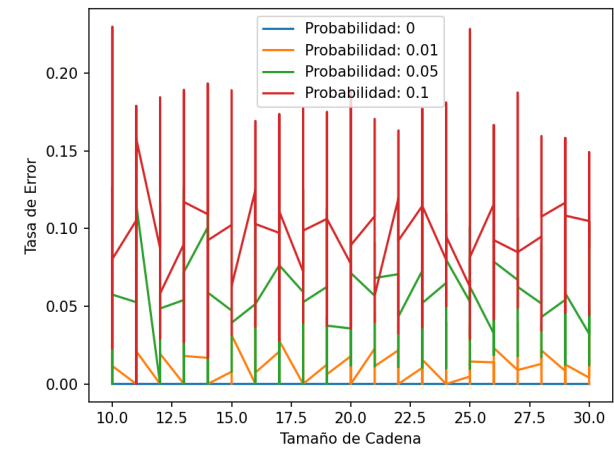
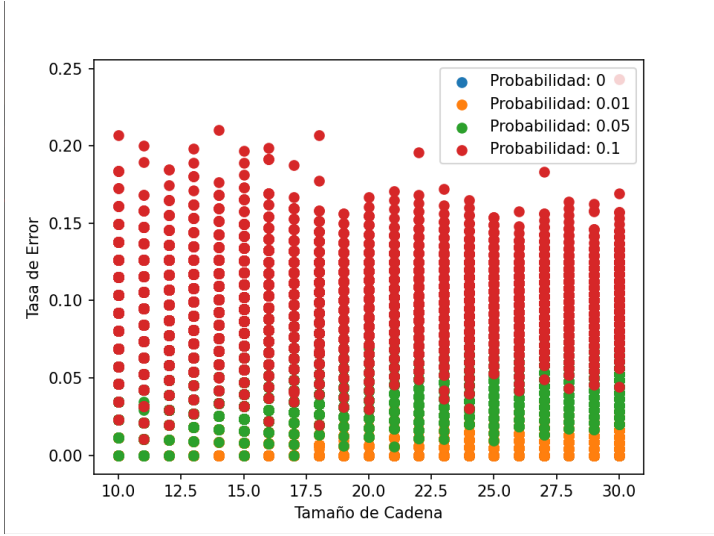
The terminal window shows the execution of a script named 'Scripts/Activate.ps1'. The user enters the message 'hola' and the system outputs the binary message '10011101100001110111101101100010100001'. The message is then encoded into a Hamming code '10011101100001110111101101100010100001'. The encoded message is sent to the receiver, but the receiver outputs a different message '10100000101011100111010000100110010110', indicating that the message was corrupted due to multiple errors.

Resultados carios con ruido 0.01



The terminal window shows the execution of a script named 'Scripts/Activate.ps1'. The user enters the message 'HOLA' and the system outputs the binary message '11001001100001110111101101100010100001'. The message is then encoded into a Hamming code '11001101100001110111101101100010100001'. The encoded message is sent to the receiver, and the receiver outputs the same message, confirming successful transmission without noise. The user then enters the message 'estas' and the system outputs the binary message '1000110101010110100110111010001110000101110011'. The message is then encoded into a Hamming code '1000110101010110100110111010001110000101110011'. The encoded message is sent to the receiver, and the receiver outputs a different message '10001101010101101001101110100011101000111010011110011', indicating that the message was corrupted due to multiple errors.

Pruebas con 10k:



Discusión

¿Qué algoritmo tuvo un mejor funcionamiento y por qué?

Considerando la frecuencia y el porcentaje de error que maneja el algoritmo de hamming podemos decir que es bastante mejor ya que logra arreglar por lo menos 1 error de los que puede causar una disminución importante en la cantidad de errores que se perciben. En general las pruebas de CRC tienden a tener un mayor error promedio.

¿Qué algoritmo es más flexible para aceptar mayores tasas de errores y por qué?

Al observar la gráfica de CRC durante la ejecución repetida del algoritmo con palabras aleatorias y probabilidades variables, se evidencia que la ocurrencia de errores en el mensaje es considerablemente mayor con una probabilidad de ruido de 0.03 en comparación con 0.2 y 0.1. No obstante, la trama suele tolerar más errores debido a la redundancia introducida por el algoritmo CRC, lo que permite detectar y gestionar los errores con mayor certeza, especialmente cuando la probabilidad de error es baja. En resumen, la combinación de la redundancia y la alta capacidad de detección del algoritmo CRC hace que la trama sea más flexible para aceptar tasas de errores más altas, asegurando una comunicación más robusta incluso en condiciones desafiantes.

¿Cuándo es mejor utilizar un algoritmo de detección de errores en lugar de uno de corrección de errores y por qué?

La elección entre un algoritmo de detección de errores y uno de corrección de errores depende de varios factores, como la importancia de los datos, la calidad de la transmisión, los recursos disponibles y el contexto de uso. A continuación, se presentan situaciones en las que es más adecuado utilizar uno u otro:

Detección de errores:

Recursos Limitados: Los algoritmos de detección de errores suelen ser más eficientes en términos de recursos computacionales que los de corrección. Si los recursos son limitados, como en sistemas con restricciones de energía o dispositivos con baja capacidad de procesamiento, la detección de errores puede ser la opción preferida.

Costo de Corrección Alto: En algunos casos, la corrección de errores puede ser costosa en términos de tiempo o recursos. Si el costo de corregir los errores es alto, y no se requiere una corrección completa, la detección puede ser una opción más eficiente.

Tolerancia a Errores Pequeños: Si la aplicación puede tolerar pequeños errores sin afectar significativamente la calidad de los datos o la interpretación, la detección de errores puede ser suficiente. Por ejemplo, en transmisiones de video o audio, pequeñas perturbaciones pueden no ser perceptibles para el usuario.

Corrección de errores:

Datos Críticos o Sensibles: Cuando los datos son críticos o sensibles, es crucial asegurarse de que los errores sean corregidos para evitar resultados incorrectos o daño a la integridad de la información. Esto es especialmente relevante en sistemas médicos, financieros o de seguridad.

Calidad de la Transmisión Baja: Si la calidad de la transmisión es baja o la probabilidad de errores es alta, la corrección de errores se vuelve más importante. Los algoritmos de corrección pueden recuperar datos dañados, mientras que la detección solo identifica los errores.

Tolerancia a Latencia: Aunque los algoritmos de corrección de errores pueden introducir cierta latencia debido al proceso de corrección, en algunas aplicaciones la tolerancia a esta latencia es posible y necesaria para garantizar la precisión de los datos.

En resumen, la elección entre detección y corrección de errores depende de la situación específica y las necesidades del sistema. La detección es más eficiente en recursos pero no corrige los errores, mientras que la corrección proporciona mayor integridad de datos pero puede requerir más recursos y tiempo.

Conclusiones

1. Al analizar la gráfica de CRC durante la ejecución repetida del algoritmo con palabras aleatorias y probabilidades variables, es evidente que la trama es flexible en la aceptación de errores. La redundancia introducida por el algoritmo CRC actúa como un "amortiguador" contra errores, permitiendo que la trama tolere más errores, especialmente cuando la probabilidad de error es baja.

2. La elección entre un algoritmo de detección de errores y uno de corrección depende del contexto de uso. Si los recursos son limitados, la detección puede ser preferible, mientras que la corrección es esencial cuando se tratan datos críticos o sensibles, o cuando la calidad de la transmisión es baja. Además, la tolerancia a errores pequeños también puede influir en la elección.
3. La importancia de los datos juega un papel clave. Los sistemas que manejan datos críticos, como sistemas médicos o financieros, requieren corrección de errores para evitar resultados incorrectos o daños a la integridad de la información.
4. Por otro lado, si el costo de corregir errores es alto o si se pueden tolerar pequeñas perturbaciones en los datos, la detección puede ser una opción más eficiente en términos de recursos.
5. La calidad de la transmisión también determina la elección. En situaciones donde la calidad es baja o la probabilidad de errores es alta, la corrección de errores es más relevante. Los algoritmos de corrección pueden recuperar datos dañados, asegurando que la información transmitida sea precisa y confiable.

Bibliografía

- and, C. (2016, February 28). Difference between CRC and Hamming Code. Computer Science Stack Exchange. <https://cs.stackexchange.com/questions/53719/difference-between-crc-and-hamming-code>
- Administrator. (2019, June 20). Error Correction and Detection Codes. ElectronicsHub. <https://www.electronicshub.org/error-correction-and-detection-codes/>
- Difference between Checksum and CRC. (2020, October 12). GeeksforGeeks; GeeksforGeeks. <https://www.geeksforgeeks.org/difference-between-checksum-and-crc/>
-