Universidad del Valle de Guatemala
Facultad de Ingeniería
Departamento de Ciencias de la Computación
Redes CC3067
Ing. Jorge Yass

Laboratorio # 2 (Parte 2)
Esquemas de detección y corrección de errores

Christopher García 20541
Maria Isabel Solano Bonilla 20504
Semestre II
Agosto 2023

Descripción de la práctica y metodología utilizada

En esta práctica, se trabajó en la implementación y experimentación de un sistema de comunicación que enfrenta los desafíos del ruido y los errores de transmisión. Estos problemas son comunes en cualquier tipo de comunicación y deben ser manejados adecuadamente para garantizar la integridad y confiabilidad de los datos transmitidos. En esta práctica, se utilizaron los algoritmos CRC-32 y Hamming para la detección y corrección de errores. Aparte de la implementación de los algoritmos, en este laboratorio lo único diferente fue la simulación de ruido en el mensaje, la codificación y decodificación del mensaje y la utilización de sockets. El ruido en este laboratorio representó un switch entre 0 y 1 de la cadena que se estaba procesando, este ruido afectaría tanto a la trama como a los bits de paridad. La codificación y decodificación del mensaje permitió ingresar un mensaje, codificarlo a ASCII binario y procesar esta información con los algoritmos, posteriormente se evaluó si el mensaje estaba correcto para poder decodificarlo, en caso contrario se corregiría la trama y se mostraría el mensaje correcto o bien se descartaría la trama.

Utilizando el laboratorio anterior se procedió a dividir los procesos en un modelo de capas, siendo esta la metodología utilizada.

Se trabajó con las siguientes capas:

- Capa de Aplicación: Se encarga de solicitar y mostrar mensajes. También gestiona la selección del algoritmo para comprobar la integridad de los datos.

- Capa de Presentación: Codifica y decodifica el mensaje en formato binario ASCII. Si no se detectan errores, decodifica el mensaje en caracteres correspondientes.

- Capa de Enlace: Calcula y verifica la integridad del mensaje utilizando el algoritmo seleccionado. También tiene la capacidad de corregir errores si el algoritmo lo permite.

- Capa de Ruido: Simula posibles interferencias aplicando ruido a la trama de datos en el emisor. Esto se hace según una probabilidad de error por bits transmitidos.

- Capa de Transmisión: Envía y recibe la trama de información a través de sockets mediante un puerto seleccionado.

Posteriormente, luego de tener un modelo completo, se procedió a realizar el apartado de pruebas que consistió en realizar, valga la redundancia, pruebas exhaustivas para evaluar el funcionamiento de los algoritmos y la arquitectura en condiciones variadas. Estas pruebas se realizan enviando y recibiendo mensajes a través del sistema implementado. Almacenando todos los resultados obtenidos continuamos al apartado de gráficas y análisis de toda esta información para determinar las conclusiones del laboratorio y del tema que se puso en práctica. Varios de estos resultados fueron almacenados en archivos csv con los cuales luego se crearon dataframes con panda para sacar estadísticas.

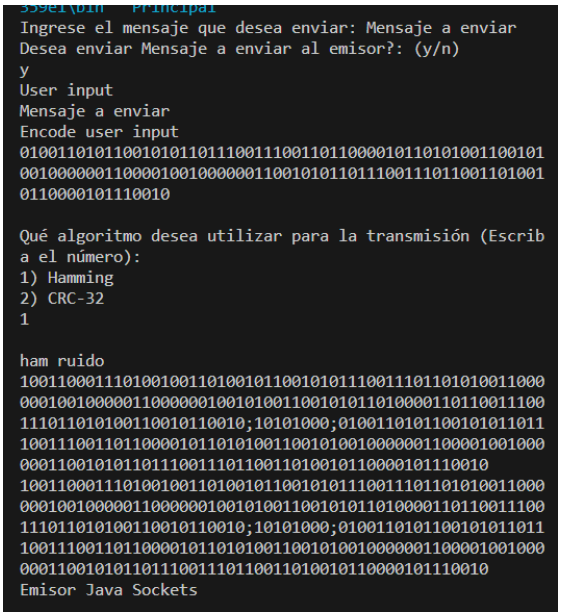
A través de esta práctica, los objetivos de aprendizaje que podemos mencionar son: cómo funcionan los modelos de capas en la comunicación, implementar algoritmos de detección y corrección de errores, experimentar con la transmisión de datos en condiciones adversas y analizar el rendimiento de los algoritmos implementados en diferentes situaciones.

Resultados

Paso simple de información

Hamming

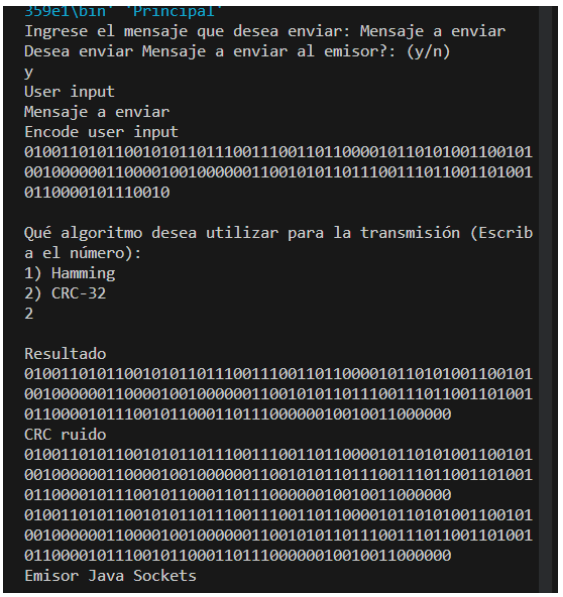
Emisor en java:



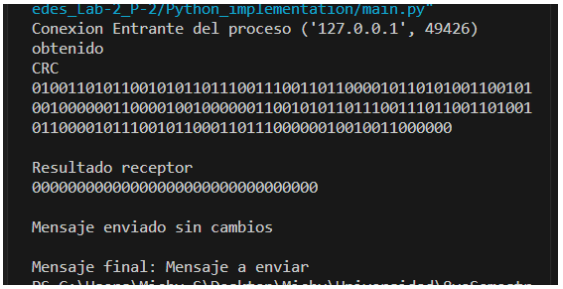
Receptor python

CRC-32, sin errores

Emisor en java:



Receptor python:



CRC-32, con errores

Emisor en java:

```
User input
Este es un mensaje que se enviar  de emisor a receptor.
Mientras m s largo, mayor es la probabilidad de cambiar
un bit
Encode user input
0100010101110011011101000110010100100000011001010111001
1001000000111010101101110001000000110110101100101011011
1001110011011000010110101001100101001000000111000101110101
0110010100100000011100110110010100100000011001010110111
0011101100110100101100001011100101010000000100000011001
0001100101001000000110010101101101011010010111001101101111
0111001000100000011000010010000000111001001100101011000
1101100101011100000111010001101111011100100010111000100000
0100110101101001011001010110111001110100011100100110000
1011100110010000001101101101000000111001100100000011011
0001100001011100100110011101101111001011000010000000110110
1011000010111100101101111011100100010000000110010101110011
0010000000110110001100001001000000111000000111001001101111
0110001001100001011000100110100101101100011010010110010
0011000010110010000100000011001000110010100100000011000
1101100001011011010110001001101001011000010111001000100000
0111010101101110001000000110001001101001011101000
```

Receptor en java:

```
Conexion Entrante del proceso ('127.0.0.1', 49446)
obtenido
CRC
0100010101110011011101000110010100100000011001010111001
1001000000111010101101110001000000110110101100101011011
1001110011011000010110101001100101001000000111000101110101
0110010100100000011100110110010100100000011001010110111
0011101100110100101100001011100101010000000100000011001
0001100101001000000110010101101101011010010111001101101111
0111001000100000011000010010000000111001001100101011000
1101100101011100000111010001101111011100100010111000100000
0100110101101001011001010110111001110100011100100110000
1011100101001000000110110110100000011100110010000000110110
0011000010111001001100111011011110010110000100000001101101
0101100001011110010110111101110010001000000011001010111001
1001000000110110100110000100100000011100000011100100110111
1101100010011000010110001001101001011011000110100101100
1000110000101100100001000000110010001100101001000000110
0011011000010110110101100010011010010110000101110010001000
0001110101011011100010000000110001001101001011101001001101
1101110001000111000001010

Resultado receptor
011010001100100010001110011111010

Mensaje enviado con errores
```

Paso masivo de información

Primero se generaron 500 tramas de un largos entre 7 y 25 bits

```java
// tramas generation
for (int i = 0; i < iterations; i ++) {

    // initialize empty array/trama
    trama = "";

    // random number between 7 and 25
    for (int j = 0; j <= rand.nextInt(25) + 7; j ++) {

        if (rand.nextBoolean()) trama += "1";
        else trama += "0";

    }

    System.out.println(trama);
    tramas.add(trama);

}
```

Emisor Hamming

eHam

	id	trama_og	codified	modified	w_noise
0	0	'00001111011'	'101010111110000;1001;00001111011'	'101010111110000;1001;00001111011'	'0'
1	1	'001111100111000'	'1100001111001110110;11010;001111100111000'	'1100001111001110110;11010;001111100111000'	'0'
2	2	'10001101101000'	'0101001001101101001;01101;10001101101000'	'0101001001101101001;01101;10001101101000'	'0'
3	3	'101010010101100'	'0101011101010010010;01110;101010010101100'	'0101011101010010010;01110;101010010101100'	'0'
4	4	'1000111111101'	'101001111111100101;10011;1000111111101'	'101001111111100101;10011;1000111111101'	'0'
...	...	...	...	...	...
495	495	'000010011011111'	'0111111010110010000;01100;000010011011111'	'0111111010110010000;01100;000010011011111'	'0'
496	496	'110110110011011010'	'0001101010110011101101;00101;110110110011011...'	'0001101010110011101101;00101;110110110011011...'	'0'
497	497	'1111001111111011100'	'01000110110111111110011111;01001;1111001111111...'	'01000110110111111110011111;01001;1111001111111...'	'0'
498	498	'00010101010'	'000110100101000;0010;00010101010'	'000110100101000;0010;00010101010'	'0'
499	499	'000001010111'	'0110110010100000;01000;000001010111'	'0110110010100000;01000;000001010111'	'0'

500 rows × 5 columns

Emisor CRC

eCRC

	id	trama_og	codified	modified	w_noise
0	0	'00001111011'	'0000111101111100010011010000100110010011100'	'0000111101111100010011010000100110010011100'	'0'
1	1	'001111100111000'	'0011111001110001000001010100101100010001000100'	'0011111001110001000001010100101100010001000100'	'0'
2	2	'10001101101000'	'1000110110100011010010000110001100010010101100'	'1000110110100011010010000101000110001001010100'	'1'
3	3	'101010010101100'	'1010100101011000111010001111010000101111011000'	'1010100101011000111010001111010000101111011000'	'0'
4	4	'1000111111101'	'10001111111011110100110000010100010001100010'	'10001111111011110100110000001010001000110010'	'1'
...	...	...	...	...	...
495	495	'000010011011111'	'0000100110111110111000110011000110111101110000'	'0000100110111110111000110011000110111101110000'	'0'
496	496	'110110110011011010'	'110110110011011010110100011110101010001000110...'	'110110110011011010101010001111010101000100011...'	'1'
497	497	'1111001111111011100'	'1111001111111011110001010011110110100011101 00...'	'111100111111101111000101001111011010001110100...'	'0'
498	498	'00010101010'	'000101010100011001010101011100110101000100'	'000101010100011001010101011100110101000100'	'0'
499	499	'000001010111'	'0000010101111100110100000001111101110010110'	'0000010101111100110100000001111101110010110'	'0'

500 rows × 5 columns

Receptor CRC

rHam

	id	alg_received	bits	reqs	orig	corr	res
0	0	'101010111110000;1001;00001111011'	'101010111110000'	'1001'	'00001111011'	'1000'	'Hubo un error en el bit 1'
1	1	'1100001111001110110;11010;001111100111000'	'1100001111001110110'	'11010'	'001111100111000'	'11110'	'Hubo un error en el bit 3'
2	2	'0101001001101101001;01101;10001101101000'	'0101001001101101001'	'01101'	'10001101101000'	'01101'	'Hubo un error en el bit 6'
3	3	'01010111010100100101;01110;101010010101100'	'01010111010100100101'	'01110'	'101010010101100'	'11110'	'Hubo un error en el bit 5'
4	4	'101001111111100101;10011;1000111111101'	'101001111111100101'	'10011'	'1000111111101'	'10011'	'Hubo un error en el bit 6'
...	...	...	...	...	...	...	...
495	495	'01111110101100100000;01100;000010011011111'	'01111110101100100000'	'01100'	'000010011011111'	'01100'	'Hubo un error en el bit 7'
496	496	'00011010101100111011011;00101;110110110011011...'	'00011010101100111011011'	'00101'	'110110110011011010'	'00111'	'Hubo un error en el bit 2'
497	497	'01000110110111111110011111;01001;1111001111111...'	'01000110110111111110011111'	'01001'	'1111001111111011100'	'11001'	'Hubo un error en el bit 5'
498	498	'000110100101000;0010;00010101010'	'000110100101000'	'0010'	'00010101010'	'0000'	'Hubo un error en el bit 2'
499	499	'0110110010100000;01000;000001010111'	'0110110010100000'	'01000'	'000001010111'	'00000'	'Hubo un error en el bit 4'

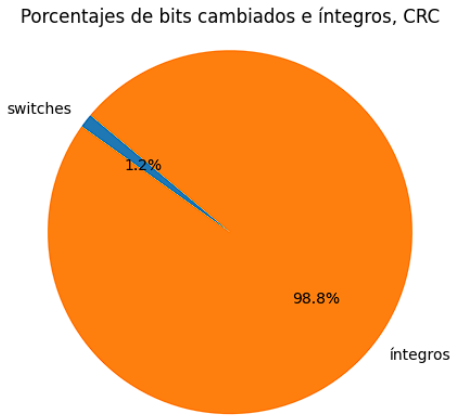
500 rows × 7 columns

rCRC

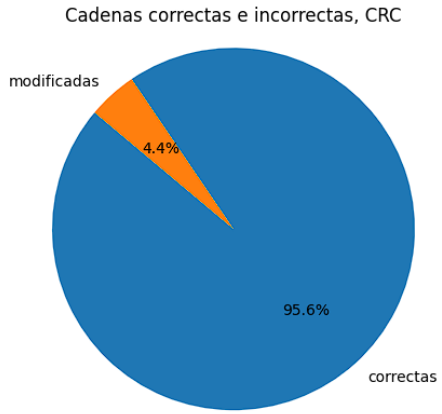
	id	alg_received	err	res
0	0	'0000111101111100010011010000100110010011100'	'0'	'00000000000000000000000000000000'
1	1	'0011111001110001000001010100101100010001000100'	'0'	'00000000000000000000000000000000'
2	2	'1000110110100011010010000101000110001001010100'	'1'	'00000000000011001010011011111100'
3	3	'1010100101011000111010001111010000101111011000'	'0'	'00000000000000000000000000000000'
4	4	'10001111111011110100110000001010001000110010'	'1'	'00000000000000111100110010100110'
...	...	...	...	...
495	495	'0000100110111110111000110011000110111101110000'	'0'	'00000000000000000000000000000000'
496	496	'110110110011011010101010001111010101000100011...'	'1'	'01111001000111111100110010101100'
497	497	'111100111111101111000101001111011010001110100...'	'0'	'00000000000000000000000000000000'
498	498	'000101010100011001010101011100110101000100'	'0'	'00000000000000000000000000000000'
499	499	'0000010101111100110100000001111101110010110'	'0'	'00000000000000000000000000000000'

500 rows × 4 columns

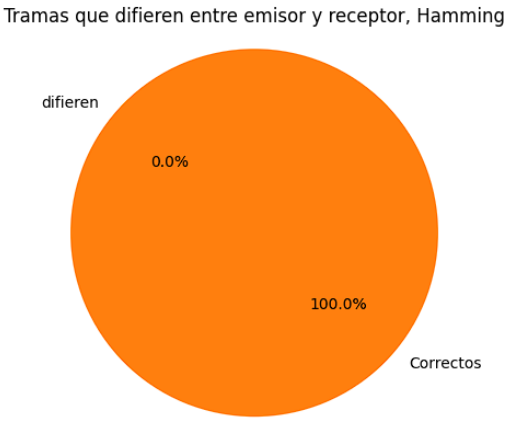
Bits que sufrieron cambios y bits íntegros en el algoritmo CRC



Tramas que sufrieron algún tipo de cambio por ruido en el algoritmo CRC



Tramas que difieren entre emisor y receptor, Hamming:



A pesar de que en el modo en el que se trabajó, los bits de paridad tenían la probabilidad de hacer switch también, en este caso, no ocurrió ningún flip que no permitiera decodificar el mensaje de manera correcta.

Discusión

Dado que en un proceso de comunicación las interferencias y errores son algo habitual este laboratorio nos permitió determinar la efectividad de diferentes algoritmos de detección y corrección de errores en este tipo de ambientes (En nuestro caso escogimos CRC-32 y Hamming). Ambos desempeñan un papel crucial en asegurar la integridad de los datos transmitidos, pero cada uno tiene sus propias limitaciones y ventajas que consideramos al momento de su implementación.

El algoritmo Hamming es conocido por su capacidad no solo para detectar errores, sino también para corregirlos. A través de la incorporación de bits de paridad, puede identificar y, en algunos casos, recuperar los bits incorrectos en los datos transmitidos. Esto lo convierte en una opción a considerar en situaciones donde la precisión y la corrección de errores son esenciales, como en aplicaciones médicas, militares, aeroespaciales, entre otras. Sin embargo, Hamming tiene sus propias limitaciones: su habilidad de corrección está limitada por la

cantidad de bits de paridad utilizados. Si los errores superan su capacidad, los datos se perderán o quedarán irrecuperables. Además, su rendimiento disminuye en condiciones de alta tasa de errores o cuando los errores están alejados entre sí en la secuencia de datos.

Por otro lado, el algoritmo CRC-32, o Cyclic Redundancy Check, se centra en la detección de errores. A través del cálculo de un valor de verificación basado en un polinomio, los datos transmitidos son verificados en el receptor para identificar cualquier error. CRC-32 destaca en la detección de una amplia gama de errores, incluidos los errores aleatorios y las ráfagas de errores. Aunque no puede corregir errores como lo hace Hamming, su capacidad de detección es robusta y efectiva en entornos con alta interferencia. Además, su implementación es rápida y eficiente, lo que lo hace adecuado para aplicaciones en tiempo real, como sistemas de almacenamiento y redes. Sin embargo, CRC-32 también tiene sus limitaciones: no puede detectar errores si los resultados del cálculo coinciden, lo que significa que es menos eficaz para errores múltiples o cercanos.

Por último, la elección entre Hamming y CRC-32 depende de las necesidades específicas de la aplicación. Si se prioriza la corrección de errores, Hamming ofrece la ventaja de la corrección de bit único, pero puede no ser tan efectivo en condiciones de alto ruido. En cambio, CRC-32 sobresale en la detección de errores y es más adecuado para aplicaciones en tiempo real, aunque carece de la capacidad de corrección.

Dados los resultados obtenidos podemos decir que, en cuanto al algoritmo con mejor funcionamiento no se puede determinar uno específico ya que uno se encuentra orientado únicamente a la detección de errores mientras que el otro además de ello busca corregir los mismos, esto genera una división y por ende el determinar cuál tiene mejor funcionamiento no es factible porque se aplicarán en diferentes escenarios. Por otra parte, al comparar su capacidad para la detección de errores vemos que CRC-32, debido a su enfoque en la detección en lugar de la corrección como ya mencionamos anteriormente, puede lidiar con una amplia gama de errores, incluidos los errores aleatorios y las ráfagas de errores. CRC-32 no intenta corregir los errores, lo que le permite ser más robusto ante fluctuaciones en la tasa de errores. Las situaciones donde se requiera uno u otro algoritmo dependerán totalmente del nivel de seguridad e integridad del mensaje que se desea, también se debe tomar en cuenta que tan crucial es la resolución de errores ya que si no es una función vital en el momento que se transmita el mensaje ya que la corrección se puede hacer posteriormente, un algoritmo como CRC-32 sería efectivo por ejemplo; y por otro lado si la corrección debe hacerse en el momento para asegurar la integridad del mensaje y que se pueda utilizar adecuadamente.

Conclusiones

- Tanto el algoritmo CRC-32 como el algoritmo Hamming, son algoritmos muy sólidos para la detección y corrección de errores, respectivamente. A pesar de que en casos muy específicos estos pueden llegar a fallar, por lo general estos realizan su trabajo de manera correcta y eficiente.
- La capa de transporte, a pesar de no ser infalible, tiene un rol muy importante al momento de la transmisión de datos ya que esta determina si el mensaje se está

enviando de manera íntegra o si es necesario hacer una corrección al mensaje o reenviarlo completamente.

- El trabajar con un modelo basado en capas permite la modularización y abstracción de sistemas sin perder la complejidad y coordinación. Cada una de las capas se encarga de una tarea específica y mediante una comunicación efectiva se logra presentar una solución a un problema que utiliza este sistema.

Bibliografía

- ¿Qué es el modelo OSI?| Ejemplos de modelos OSI. (2023). Cloudflare. https://www.cloudflare.com/es-es/learning/ddos/glossary/open-systems-interconnection-model-osi/
- Ramón Invarato. (2016, October 5). Código de Hamming: Detección y Corrección de errores - Jarroba. Jarroba. https://jarroba.com/codigo-de-hamming-deteccion-y-correccion-de-errores/
- Command Understanding Cyclic Redundancy Check. (2021). Archive.org. https://web.archive.org/web/20050408104838/http://www.4d.com/docs/CMU/CMU79909.HTM