UNIVERSIDAD DEL VALLE DE GUATEMALA
Redes
Sección 21
Jorge Yass

Laboratorio 2

Parte 2

Oscar Esteban Donis Martínez - 21610
Abner Iván García Alegría - 21285

Link repositorio Github:

https://github.com/estebandonis/CRC32-Hamming7-4-Networks.git

Descripción de la práctica

Para esta práctica de laboratorio debíamos de comunicar el emisor con el receptor de forma automática lo cual ya no era de forma manual, lo cual se debía de hacer mediante la herramienta de sockets para poder comunicar el emisor con el receptor de ambos lenguajes y al final se deben de comparar para determinar cuál es el algoritmo que lo hace más rápido, básicamente los pros y los contras de ambos algoritmos para determinar cuál es el mejor. Nuestros algoritmos utilizados para esta práctica fueron: el algoritmo hamming y el algoritmo CRC-32.

Resultados

Mensajes con probabilidad de 0% a 25%

Algoritmo Hamming:

Mensaje 1: hola con 0%

```
C:\Users\Personal\Documents\Universidad\8 Semestre\Redes\Laboratorio2-Redes\Hamming>.\Emisor
Ingrese el mensaje: hola
Mensaje codificado en Hamming (en bits): 011001001010011001111110011001100000011000000110
Ingrese la probabilidad de error (ej. 0.01 para 1%): 0.00

Inicializando Winsock...Inicializado.
Socket creado.
Conectado al servidor.
Mensaje enviado.
```

```
Escuchando en el puerto 65432...
Conectado por ('127.0.0.1', 64106)
Mensaje recibido en bits: 011001001010011001111110011001100000011000000110

Bloques de 12 bits recibidos:
Bloque 1: 011001001010
Bloque 2: 011001111110
Bloque 3: 011001100000
Bloque 4: 011000000110
```

```
Mensaje recibido y sin errores.

Mensaje en bits concatenados: 011010000110111101101100001100001
Mensaje decodificado: hola
```

Mensaje 2: solo con 20%

```
Ingrese el mensaje: solo
Mensaje codificado en Hamming (en bits): 011110011110011111100110000001100110011111100110
Ingrese la probabilidad de error (ej. 0.01 para 1%): 0.2

Inicializando Winsock...Inicializado.
Socket creado.
Conectado al servidor.
Mensaje enviado.
```

```
thon.exe" "c:/Users/Personal/Documents/Universidad/8 Semestre/Redes/Laboratorio2-Redes/Hamming/Receptor.py"
Escuchando en el puerto 65432...
Conectado por ('127.0.0.1', 64665)
Mensaje recibido en bits: 010100111100011111100011001010000111011111101110

Bloques de 12 bits recibidos:
Bloque 1: 010101111100
Bloque 2: 001111100011
Bloque 3: 011010000111
Bloque 4: 011011101110
```

```
Bits de paridad:
Bloque 1 - Bits de paridad: 0101
Bloque 2 - Bits de paridad: 0011
Bloque 3 - Bits de paridad: 0110
Bloque 4 - Bits de paridad: 0110
Mensaje recibido y corregido.

Mensaje en bits concatenados: 010101110111110000100000101111101
Hubo más de dos errores lo cual no se pudo corregir.
```

Solo con 0.00001%

```
C:\Users\Personal\Documents\Universidad\8 Semestre\Redes\Laboratorio2-Redes\Hamming>.\Emisor
Ingrese el mensaje: solo
Mensaje codificado en Hamming (en bits): 011110011110011001111110011001100000011001111110
Ingrese la probabilidad de error (ej. 0.01 para 1%): 0.00001

Inicializando Winsock...Inicializado.
Socket creado.
Conectado al servidor.
Mensaje enviado.
```

```
PS C:\Users\Personal\Documents\Universidad\8 Semestre\Redes\Laboratorio2-Redes\Hamming> & "C:/Program Files/Python311/py
thon.exe" "c:/Users/Personal/Documents/Universidad/8 Semestre/Redes/Laboratorio2-Redes/Hamming/Receptor.py"
Escuchando en el puerto 65432...
Conectado por ('127.0.0.1', 64984)
Mensaje recibido en bits: 011110011110011001111110011001100000011001111110

Bloques de 12 bits recibidos:
Bloque 1: 011110011110
Bloque 2: 011001111110
Bloque 3: 011001100000
Bloque 4: 011001111110
```

```
Bits de paridad:
Bloque 1 - Bits de paridad: 0111
Bloque 2 - Bits de paridad: 0110
Bloque 3 - Bits de paridad: 0110
Bloque 4 - Bits de paridad: 0110
Mensaje recibido y sin errores.

Mensaje en bits concatenados: 011100110110111101101100011011111
Mensaje decodificado: solo
PS C:\Users\Personal\Documents\Universidad\8 Semestre\Redes\Laboratorio2-Redes\Hamming>
```

Algoritmo CRC-32:

Mensaje 1: Hola con 0%

```
Si quiere salir, escriba 'exit'
Ingrese el mensaje: Hola
Mensaje:  Hola
Enter the probability of noise (0-1):
Enter dividend: 0
Enter divisor: 1
Received from server: Message received correctly
Mensaje enviado:  01001000011011110110110001100001 01101000111001000111100101100011
```

```
Message received: 01001000011011110110110001100001 01101000111001000111100101100011
Message received has no errors.
Message decoded: Hola
```

Mensaje 2: Solo con 20%

```
Ingrese el mensaje: Solo
Mensaje:  Solo
Enter the probability of noise (0-1):
Enter dividend: 1
Enter divisor: 5
Received from server: Message received correctly
Mensaje enviado:  01000001010000110100100001101111 00111110001000100100101111011101
```

```
Message received: 11010000110000110110110001001111 00111110101010111110001100111001
Error: Found errors during verification.
```

Solo con 0.00001%

```
Ingrese el mensaje: Solo
Mensaje:  Solo
Enter the probability of noise (0-1):
Enter dividend: 1
Enter divisor: 100000
Received from server: Message received correctly
Mensaje enviado:  01010011011011110110110001101111 00111110001000101100001100011001
```

```
Message received: 01010011011011110110110001101111 00111110001000101100001100011001
Message received has no errors.
Message decoded: Solo
```

Mensajes con probabilidad del 26% a 50%

Algoritmo Hamming:

Mensaje 1: universidad con 30%

```
C:\Users\Personal\Documents\Universidad\8 Semestre\Redes\Laboratorio2-Redes\Hamming>.\Emisor
Ingrese el mensaje: universidad
Mensaje codificado en Hamming (en bits): 1011010111101001111001101011001001101100110111100011010001101001100111100111001
111101011001001101101010001100110000001101101010000110
Ingrese la probabilidad de error (ej. 0.01 para 1%): 0.3

Inicializando Winsock...Inicializado.
Socket creado.
Conectado al servidor.
Mensaje enviado.
```

```
Escuchando en el puerto 65432...
Conectado por ('127.0.0.1', 64743)
Mensaje recibido en bits: 1010011001100011111110101011010001101101110111000110101001000011000011100011110111111111011001111010000011101111000110010110000110
Error: Posición de error (14) fuera de rango.
Error: Posición de error (14) fuera de rango.

Bloques de 12 bits recibidos:
Bloque 1: 101000100110
Bloque 2: 001111110010
Bloque 3: 101101001110
Bloque 4: 010111011110
Bloque 5: 000101010010
Bloque 6: 100110000111
```

```
Bloque 7: 000011101111
Bloque 8: 111110110010
Bloque 9: 110100000111
Bloque 10: 011110101100
Bloque 11: 101100000110

Bits de paridad:
Bloque 1 - Bits de paridad: 1010
Bloque 2 - Bits de paridad: 0011
Bloque 3 - Bits de paridad: 1011
Bloque 4 - Bits de paridad: 0101
Bloque 5 - Bits de paridad: 0001
Bloque 6 - Bits de paridad: 1001
Bloque 7 - Bits de paridad: 0000
```

```
Bloque 7 - Bits de paridad: 0000
Bloque 8 - Bits de paridad: 1111
Bloque 9 - Bits de paridad: 1101
Bloque 10 - Bits de paridad: 0111
Bloque 11 - Bits de paridad: 1011
Mensaje recibido y corregido.

Mensaje en bits concatenados: 10101101001111101011001110110110011101000010001000111011111011011010001011100011011000 1
Hubo más de dos errores lo cual no se pudo corregir.
```

Mensaje 2: probabilidad con 45%

```
C:\Users\Personal\Documents\Universidad\8 Semestre\Redes\Laboratorio2-Redes\Hamming>.\Emisor
Ingrese el mensaje: probabilidad
Mensaje codificado en Hamming (en bits): 000000011110100110011110011111100110000110000110011000000110000110000110101100100110000001100110101100100110110101000110011000000110110101000110
Ingrese la probabilidad de error (ej. 0.01 para 1%): 0.45

Inicializando Winsock...Inicializado.
Socket creado.
Conectado al servidor.
Mensaje enviado.
```

```
thon.exe" "c:/Users/Personal/Documents/Universidad/8 Semestre/Redes/Laboratorio2-Redes/Hamming/Receptor.py"
Escuchando en el puerto 65432...
Conectado por ('127.0.0.1', 64764)
Mensaje recibido en bits: 0110011000011000101001010111101010111101010011001110010111011001010011110001000000111100001100
1010000110011011110000110101100100110110100001101110
```

```
Bloques de 12 bits recibidos:
Bloque 1: 111001100001
Bloque 2: 100010100101
Bloque 3: 111110101011
Bloque 4: 110101001100
Bloque 5: 111001010101
Bloque 6: 100100001111
Bloque 7: 000100000111
Bloque 8: 110001110010
Bloque 9: 100001100111
Bloque 10: 101100001101
Bloque 11: 011001001101
Bloque 12: 100001111110
```

```
Bloque 5 - Bits de paridad: 1110
Bloque 6 - Bits de paridad: 1001
Bloque 7 - Bits de paridad: 0001
Bloque 8 - Bits de paridad: 1100
Bloque 9 - Bits de paridad: 1000
Bloque 10 - Bits de paridad: 1011
Bloque 11 - Bits de paridad: 0110
Bloque 12 - Bits de paridad: 1000
Mensaje recibido y corregido.

Mensaje en bits concatenados: 011011001000010101110100110110011111010111001100100010000110001101000110111110001011010011
001101
Hubo más de dos errores lo cual no se pudo corregir.
```

Algoritmo CRC-32:

Mensaje 1: universidad con 30%

```
Ingrese el mensaje: universidad
Mensaje:  universidad
Enter the probability of noise (0-1):
Enter dividend: 30
Enter divisor: 100
Received from server: Message received correctly
Mensaje enviado:  10000010111001000001110001101101 10111111000111011110110001000110 11111110010101011111101001110000 011111101111
01000110100110101101 01011011000001001100000001110100 01010100100111011110001010100101
```

```
Message received: 10000010111001000001110001101101 10111111000111011110110001000110 11111110010101011111101001
01110000 01111110111101000110100110101101 01011011000001001100000001110100 01010100100111011110001010100101
Error: Found errors during verification.
```

Mensaje 2: probabilidad con 45%

```
Ingrese el mensaje: probabilidad
Mensaje:  probabilidad
Enter the probability of noise (0-1):
Enter dividend: 45
Enter divisor: 100
Received from server: Message received correctly
Mensaje enviado:  11110101001000100110100110000010 11100010101001000000110010101110 10101110000001010011000110101000 000010101110
01110011101111001101 01100111111101010111110001100101 10010100110101011110100110010010
```

```
Message received: 11110101001000100110100110000010 11100010101001000000110010101110 10101110000001010011000
10101000 00001010111001110011101111001101 01100111111101010111110001100101 10010100110101011110100110010010
Error: Found errors during verification.
```

Mensajes con probabilidad del 50% al 80%

Algoritmo Hamming:

Mensaje 1: redes con 60%

```
C:\Users\Personal\Documents\Universidad\8 Semestre\Redes\Laboratorio2-Redes\Hamming>.\Emisor
Ingrese el mensaje: redes
Mensaje codificado en Hamming (en bits): 100110011110001101000110110101000110001101000110011110011110
Ingrese la probabilidad de error (ej. 0.01 para 1%): 0.60

Inicializando Winsock...Inicializado.
Socket creado.
Conectado al servidor.
Mensaje enviado.
```

```
thon.exe" "c:/Users/Personal/Documents/Universidad/8 Semestre/Redes/Laboratorio2-Redes/Hamming/Receptor.py"
Escuchando en el puerto 65432...
Conectado por ('127.0.0.1', 64813)
Mensaje recibido en bits: 010101000111010010000000101010101010010001011101001011001011
```

```
Bloques de 12 bits recibidos:
Bloque 1: 010111000111
Bloque 2: 011010000000
Bloque 3: 101010101010
Bloque 4: 010001011101
Bloque 5: 011011001011

Bits de paridad:
Bloque 1 - Bits de paridad: 0101
Bloque 2 - Bits de paridad: 0110
Bloque 3 - Bits de paridad: 1010
Bloque 4 - Bits de paridad: 0100
Bloque 5 - Bits de paridad: 0110
```

```
Mensaje en bits concatenados: 010110010100000010100100010010110010100 0
Hubo más de dos errores lo cual no se pudo corregir.
```

Mensaje 2: compiladores con 80%

```
C:\Users\Personal\Documents\Universidad\8 Semestre\Redes\Laboratorio2-Redes\Hamming>.\Emisor
Ingrese el mensaje: compiladores
Mensaje codificado en Hamming (en bits): 111110000110011111100110111001100110000000011110101100100110000001100110011000
0011011010100011001111110011010011001111000110100011001111001110
Ingrese la probabilidad de error (ej. 0.01 para 1%): 0.8

Inicializando Winsock...Inicializado.
Socket creado.
Conectado al servidor.
Mensaje enviado.
```

```
PS C:\Users\Personal\Documents\Universidad\8 Semestre\Redes\Laboratorio2-Redes\Hamming> & "C:/Program Files/Python311/py
thon.exe" "c:/Users/Personal/Documents/Universidad/8 Semestre/Redes/Laboratorio2-Redes/Hamming/Receptor.py"
Escuchando en el puerto 65432...
Conectado por ('127.0.0.1', 64884)
Mensaje recibido en bits: 000000011111100000001001000110011001111111100001000011011001111110011101101111110001010010100
0100000001101101111101111110010111101100001100111
```

```
Bloques de 12 bits recibidos:
Bloque 1: 100000011111
Bloque 2: 100000000001
Bloque 3: 100110011001
Bloque 4: 111111100000
Bloque 5: 000011011001
Bloque 6: 111110011101
Bloque 7: 111111110001
Bloque 8: 010110100001
Bloque 9: 010000011011
Bloque 10: 011101101111
Bloque 11: 110010111001
Bloque 12: 100001100111
```

```
Bloque 4 - Bits de paridad: 1111
Bloque 5 - Bits de paridad: 0000
Bloque 6 - Bits de paridad: 1111
Bloque 7 - Bits de paridad: 1111
Bloque 8 - Bits de paridad: 0101
Bloque 9 - Bits de paridad: 0100
Bloque 10 - Bits de paridad: 0111
Bloque 11 - Bits de paridad: 1100
Bloque 12 - Bits de paridad: 1000
Mensaje recibido y corregido.

Mensaje en bits concatenados: 000000111000000000010010111111000000101011110011101111100100010000000010011111011100011110
001101
Hubo más de dos errores lo cual no se pudo corregir.
```

Algoritmo CRC-32:

Mensaje 1: redes con 60%

```
Ingrese el mensaje: redes
Mensaje:  redes
Enter the probability of noise (0-1):
Enter dividend: 60
Enter divisor: 100
Received from server: Message received correctly
Mensaje enviado:  10000100110111101001111010001000 10010111110110000101001000010001 11011000100010001001101111001010 101010100000
10000101011011111101
```

```
Message received: 10000100110111101001111010001000 10010111110110000101001000010001 11011000100010001001101111001010
Error: Found errors during verification.
```

Mensaje 2: compiladores con 80%

```
Ingrese el mensaje: compiladores
Mensaje:  compiladores
Enter the probability of noise (0-1):
Enter dividend: 80
Enter divisor: 100
Received from server: Message received correctly
Mensaje enviado:  10011010100010001111001010011110 00000110100010101110110100010110 00110111100100111101101010001010 101011011110
01110111111010111000 10000010100011011001001011001100 01000101100001101111001111110011
```

```
Message received: 10011010100010001111001010011110 00000110100010101110110100010110 00110111100100111101101010001010
Error: Found errors during verification.
```

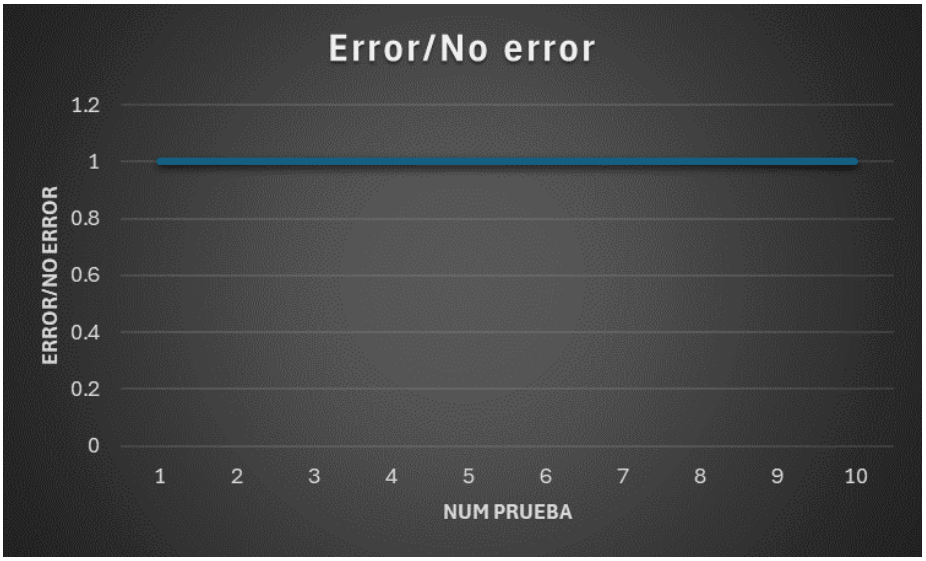
Gráficos

0 = Dio errores

1 = No dio errores

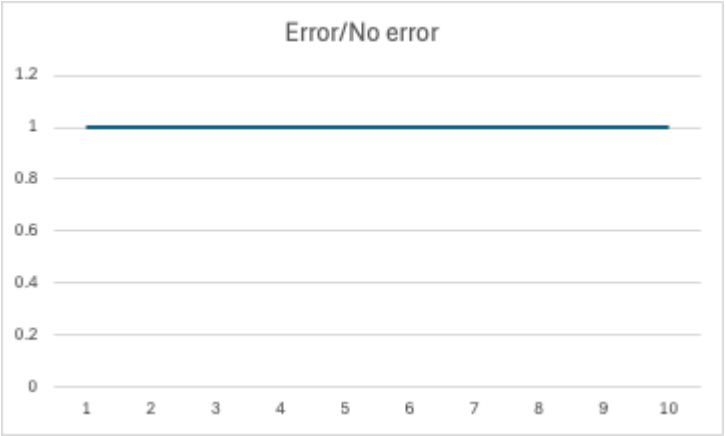
Algoritmo Hamming: hola con 0%

Num de prueba	Error/No error
1	1
2	1
3	1
4	1
5	1
6	1
7	1
8	1
9	1
10	1
	Accuracy = 100%



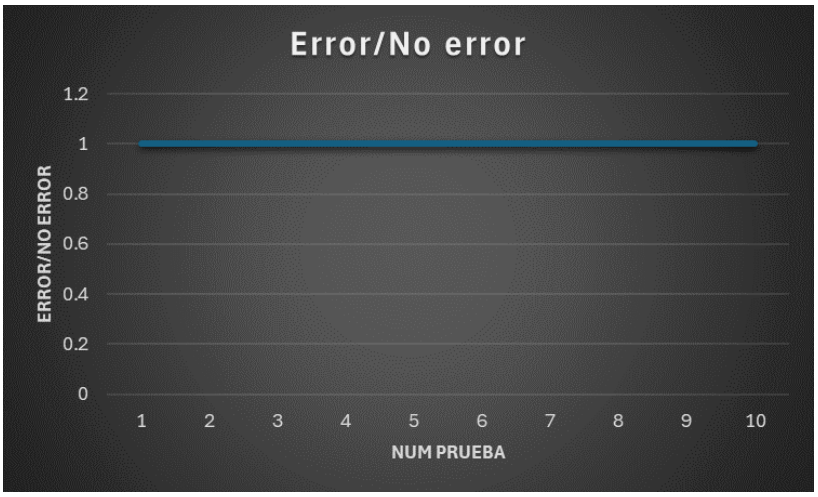
CRC-32: hola con 0%

Num de prueba	Error/No error	Cantidad errores
1	1	0
2	1	0
3	1	0
4	1	0
5	1	0
6	1	0
7	1	0
8	1	0
9	1	0
10	1	0
	Accuracy = 100%	



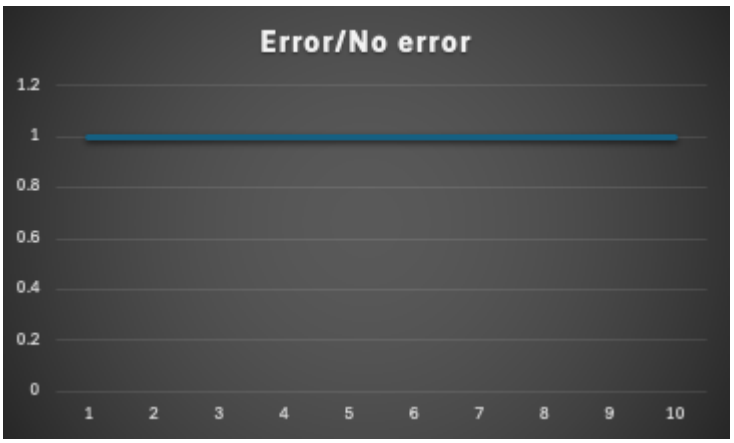
Algoritmo Hamming: solo con 0.00001%

Num de prueba	Error/No error
1	1
2	1
3	1
4	1
5	1
6	1
7	1
8	1
9	1
10	1



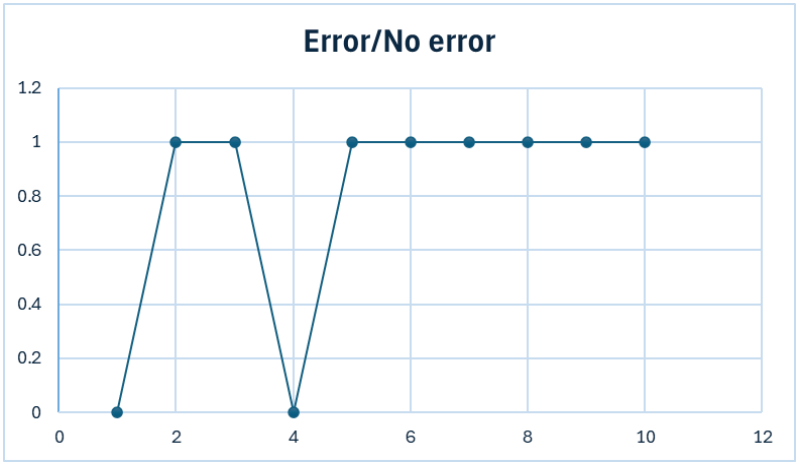
CRC-32: solo con 0.00001%

Num de prueba	Error/No error	Cantidad errores
1	1	0
2	1	0
3	1	0
4	1	0
5	1	0
6	1	0
7	1	0
8	1	0
9	1	0
10	1	0
	Accuracy = 100%	



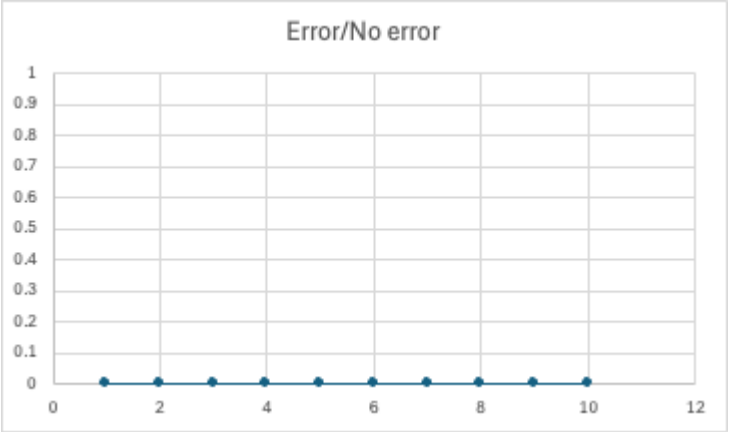
Algoritmo Hamming: universidad con 30%

Num de prueba	Error/No error
1	0
2	1
3	1
4	0
5	1
6	1
7	1
8	1
9	1
10	1
	Accuracy = 80%



CRC-32: universidad con 30%

Num de prueba	Error/No error	Cantidad errores
1	0	63
2	0	61
3	0	60
4	0	51
5	0	53
6	0	62
7	0	52
8	0	53
9	0	59
10	0	56
	Accuracy = 100%	57

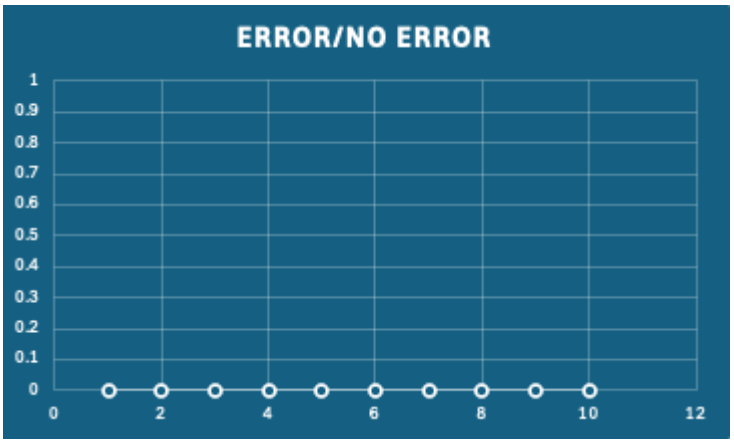


Algoritmo Hamming: redes con 60%

Num de prueba	Error/No error
1	0
2	0
3	0
4	0
5	0
6	1
7	0
8	0
9	1
10	0
	Accuracy = 80%

CRC-32: redes con 60%

Num de prueba	Error/No error	Cantidad errores
1	0	85
2	0	77
3	0	74
4	0	70
5	0	66
6	0	68
7	0	83
8	0	83
9	0	76
10	0	72
	Accuracy = 100%	75.4



R// Cómo podemos ver en las gráficas el algoritmo hamming al realizar dichas pruebas vemos que mientras menor sea el porcentaje del ruido menos errores tendrá, mientras que el porcentaje aumente mayor errores tendrá, al realizar las pruebas me di cuenta que aunque el porcentaje es alto mucho más se va confundiendo por lo cual eso es una desventaja de este algoritmo ya que no es capaz de resolver más de dos errores por lo cual lo hace útil para situaciones en donde se requiera de un error en este caso porcentajes muy bajos mientras que no es muy bueno en ocasiones donde hay más de dos errores en las

pruebas. En CRC-32 podemos observar que nunca se confundió en los test que realizamos, incluso teniendo alta o baja probabilidad de error, podemos observar que este algoritmo es bastante preciso al momento de detectar errores. Esto se puede deber al grado del polinomio que se utiliza para realizar la verificación, ya que, a mayor grado el polinomio, más robusta es la verificación que se le realiza. Esto, aunque es beneficioso, también implica muchas más operaciones para realizar la verificación, por lo que o equiparamos el poder computacional para realizar dichas verificaciones o tendremos que esperar mucho más tiempo para finalmente recibir el mensaje enviado.

Discusión

Al realizar las pruebas vimos que Hamming es apreciado por su simplicidad y rapidez en la corrección de errores de un solo bit, siendo ideal para sistemas embebidos con recursos limitados ya que no es capaz de poder corregir más de un error ya al realizar las pruebas aplicando ruido vimos que a la hora de cambiar bits más de dos errores no es capaz de corregirlos lo cual muestra a veces mensajes que no son los correctos un dato interesante que se observó sobre hamming es que se puede alterar los bits de tal manera que el algoritmo no sea capaz de detectar errores es decir que diga que no hubo error pero muestra otro mensaje lo cual es muy interesante ya que este algoritmo tiene sus pros y sus contras. Por otro lado, CRC32 nos llamó la atención por su robustez en la detección de múltiples errores, siendo crucial para aplicaciones que requieren alta integridad de datos, como las redes de comunicación y el almacenamiento. Se mencionó que, aunque CRC-32 es más complejo y requiere más recursos, su capacidad para detectar errores sin necesidad de retransmisión lo hace superior en muchos casos.

Comentario grupal sobre el tema

Esta actividad de los sockets fue enriquecedora, ya que nos permitió poder aprender sobre la herramienta de sockets ya que nos fue interesante como es que tanto el emisor como el receptor se comunican mediante esta herramienta, es super emocionante como es que funciona esto de mandar los mensajes de un lado a otro. Además nos llamó la atención cómo es que los algoritmos son capaces de detectar errores y en caso del algoritmo de hamming de corregir un bits y mandar el mensaje correcto que el emisor mando es increíble al igual con el tema de las capas nos sirvió mucho para esta práctica.

Conclusiones

- **Robustez y Simplicidad:** Hamming es más sencillo de implementar y eficiente en términos de velocidad, pero está limitado a la corrección de errores de un solo bit. CRC-32, por otro lado, proporciona una mayor robustez en la detección de errores, siendo capaz de identificar múltiples errores, aunque su implementación es más compleja y requiere más recursos computacionales.
- **Aplicaciones adecuadas:** Hamming es adecuado para aplicaciones donde los errores simples son más comunes y los recursos son limitados, como en sistemas embebidos. CRC-32 es más adecuado para aplicaciones que requieren una alta integridad de datos, como en redes de comunicación y sistemas de almacenamiento.
- **Eficiencia en la Implementación:** Hamming es más eficiente en términos de implementación y procesamiento, lo que lo hace ideal para dispositivos con capacidades limitadas de procesamiento y memoria. En cambio CRC-32 es más

complejo de implementar ya que requiere de una extensa cantidad de XOR debido a su gran capacidad de tamaño de los bits.

Citas y Referencias

1. Invarato, R. (2017, 12 agosto). Código de Hamming: Detección y Corrección de errores - Jarroba. Jarroba. https://jarroba.com/codigo-de-hamming-deteccion-y-correccion-de-errores/.
2. *CRC32: Verificación de redundancia cíclica*. (s. f.). https://www.jc-mouse.net/java/crc32-verificacion-de-redundancia-ciclica.
3. *Hamming y CRC*. (2009, 14 octubre). [Diapositivas]. SlideShare. https://es.slideshare.net/slideshow/hamming-y-crc/2226045.