UVG UNIVERSIDAD DEL VALLE DE GUATEMALA

CC3067 - Redes
Sección 10

Pablo Andrés Zamora Vásquez - 21780 Diego Andrés Morales Aquino - 21762

Laboratorio No. 2
Esquemas de detección y corrección de errores

Link al repositorio de GitHub: https://github.com/pabloozamora/UVG_Redes_Lab_2.git

Implementación de algoritmos

Algoritmo para corrección de errores

Se decidió implementar códigos de Hamming:

- Emisor (Codificación): Python
- Receptor (Decodificación): Javascript

Algoritmo para identificación de errores

Se decidió implementar Fletcher *checksum*:

- Emisor (Codificación): Python
- Receptor (Decodificación): Javascript

Escenarios de pruebas

Sin errores

- Cadena a codificar: 1001001010

Hamming

Emisor:

```
PS D:\2024\Semestre_2\Redes\lab2> & "C:/Users/Pablo
o/hamming/hamming_encoder.py
Ingrese la cadena de bits a codificar: 1001001010
Datos codificados: 01100010001010
PS D:\2024\Semestre_2\Redes\lab2>
```

Mensaje codificado: 01100010001010

Receptor:

```
PS D:\2024\Semestre_2\Redes\lab2\repo\hamming> node hamming_decoder.js
Ingrese la cadena de bits codificada: 01100010001010
No se detectaron errores en la secuencia codificada.
Datos decodificados: 1001001010
PS D:\2024\Semestre_2\Redes\lab2\repo\hamming>
```

Fletcher16 checksum

Emisor:

```
PS D:\diego\OneDrive - UVG\Documentos Universidad\Semestr
Ingresar mensaje en binario a codificar: 1001001010
Mensaje completo codificado:
1001001010010011100l001100
PS D:\diego\OneDrive - UVG\Documentos Universidad\Semestr
```

Mensaje codificado: 1001001010010011100l001100

Receptor:

```
PS D:\diego\OneDrive - UVG\Documentos Universidad\Semestre 8\re
iego/OneDrive - UVG/Documentos Universidad/Semestre 8/redes/lab
Ingresar cadena codificada: 1001001010010011100l001100
Mensaje correcto! El mensaje es: 1001001010
PS D:\diego\OneDrive - UVG\Documentos Universidad\Semestre 8\re
```

- Cadena a codificar: 0101101

Hamming

Emisor:

```
PS D:\2024\Semestre_2\Redes\lab2> & "C:/Users/Pablo
o/hamming/hamming_encoder.py
Ingrese la cadena de bits a codificar: 0101101
Datos codificados: 00001010101
PS D:\2024\Semestre_2\Redes\lab2>
```

Mensaje codificado: 00001010101

Receptor:

```
PS D:\2024\Semestre_2\Redes\lab2\repo\hamming> node hamming_decoder.js
Ingrese la cadena de bits codificada: 00001010101
No se detectaron errores en la secuencia codificada.
Datos decodificados: 0101101
PS D:\2024\Semestre_2\Redes\lab2\repo\hamming>
```

Fletcher16 checksum

Emisor:

```
PS D:\diego\OneDrive - UVG\Documentos Universidad\Semest
Ingresar mensaje en binario a codificar: 0101101
Mensaje completo codificado:
010110100101101001011O1
PS D:\diego\OneDrive - UVG\Documentos Universidad\Semest
```

Mensaje codificado: 0101101001011010010110 1

Receptor:

```
PS D:\diego\OneDrive - UVG\Documentos Universidad\Semestre
iego/OneDrive - UVG/Documentos Universidad/Semestre 8/rede
Ingresar cadena codificada: 0101101001011010010110 1
Mensaje correcto! El mensaje es: 0101101
PS D:\diego\OneDrive - UVG\Documentos Universidad\Semestre
```

- Mesaje a codificar: 0011010101111001

Hamming

Emisor:

```
PS D:\2024\Semestre_2\Redes\lab2> & "C:/Users/Pablo Zamora/
o/hamming/hamming_encoder.py
Ingrese la cadena de bits a codificar: 0011010101111001
Datos codificados: 00000110010101111 1001
PS D:\2024\Semestre_2\Redes\lab2>
```

Mensaje codificado: 00000110010101111 1001

Receptor:

```
PS D:\2024\Semestre_2\Redes\lab2\repo\hamming> node hamming_decoder.js
Ingrese la cadena de bits codificada: 000001100101011111001
No se detectaron errores en la secuencia codificada.
Datos decodificados: 0011010101111001
PS D:\2024\Semestre_2\Redes\lab2\repo\hamming>
```

Fletcher16 checksum

Emisor:

```
PS D:\diego\OneDrive - UVG\Documentos Universidad\Semestre 8\r
Ingresar mensaje en binario a codificar: 0011010101111001
Mensaje completo codificado:
00110101011110011110001110101110
PS D:\diego\OneDrive - UVG\Documentos Universidad\Semestre 8\r
```

Mensaje codificado: 00110101011110011110001110101110

Receptor:

```
PS D:\diego\OneDrive - UVG\Documentos Universidad\Semestre 8\red
iego/OneDrive - UVG/Documentos Universidad/Semestre 8/redes/labs
Ingresar cadena codificada: 00110101011110011110001110101110
Mensaje correcto! El mensaje es: 0011010101111001
PS D:\diego\OneDrive - UVG\Documentos Universidad\Semestre 8\red
```

Un error

- Cadena a codificar: 110

Hamming

Emisor:

```
PS D:\2024\Semestre_2\Redes\lab2> & "C:/Users/Pablo
o/hamming/hamming_encoder.py
Ingrese la cadena de bits a codificar: 110
Datos codificados: 011110
PS D:\2024\Semestre_2\Redes\lab2>
```

Mensaje codificado: 011110

Receptor:

Se introduce un error en la posición 5: 0111**0**0

```
PS D:\2024\Semestre_2\Redes\lab2\repo\hamming> node hamming_decoder.js
Ingrese la cadena de bits codificada: 011110
Datos codificados con error introducido: 011100
Bit incorrecto en la posición: 5. Corrigiendo...
Datos decodificados: 110
PS D:\2024\Semestre_2\Redes\lab2\repo\hamming>
```

Fletcher16 checksum

Emisor:

```
PS D:\diego\OneDrive - UVG\Documentos Universidad\Sem
Ingresar mensaje en binario a codificar: 110
Mensaje completo codificado:
1100000011000000110
PS D:\diego\OneDrive - UVG\Documentos Universidad\Sem
```

Mensaje codificado: 1100000011000000110

Receptor:

Se introduce un error en la posición 2: 1**1**000000011000000110

```
PS D:\diego\OneDrive - UVG\Documentos Universidad\Semestre 8
iego/OneDrive - UVG/Documentos Universidad/Semestre 8/redes/
Ingresar cadena codificada: 1000000011000000110
Checksum incorrecto. El mensaje contiene errores.
PS D:\diego\OneDrive - UVG\Documentos Universidad\Semestre 8
```

- Cadena a codificar: 110110101

Hamming

Emisor:

```
PS D:\2024\Semestre_2\Redes\lab2> & "C:/Users/Pablo
o/hamming/hamming_encoder.py
Ingrese la cadena de bits a codificar: 110110101
Datos codificados: 0111101110101
PS D:\2024\Semestre_2\Redes\lab2>
```

Mensaje codificado: 0111101110101

Receptor:

Se introduce un error en la posición 7: 011110**0**110101

```
PS D:\2024\Semestre_2\Redes\lab2\repo\hamming> node hamming_decoder.js
Ingrese la cadena de bits codificada: 0111101110101
Datos codificados con error introducido: 0111100110101
Bit incorrecto en la posición: 7. Corrigiendo...
Datos decodificados: 110110101
PS D:\2024\Semestre_2\Redes\lab2\repo\hamming>
```

Fletcher16 checksum

Emisor:

```
PS D:\diego\OneDrive - UVG\Documentos Universidad\Seme
Ingresar mensaje en binario a codificar: 110110101
Mensaje completo codificado:
1101101011011011110110110
PS D:\diego\OneDrive - UVG\Documentos Universidad\Seme
```

Mensaje codificado: 11011010110110111110110110

Receptor:

Se introduce un error en la posición 13: 1101101011011011110110110

```
PS D:\diego\OneDrive - UVG\Documentos Universidad\Semest
iego/OneDrive - UVG/Documentos Universidad/Semestre 8/re
Ingresar cadena codificada: 11011010110100111110110110
Checksum incorrecto. El mensaje contiene errores.
PS D:\diego\OneDrive - UVG\Documentos Universidad\Semest
```

- Cadena a codificar: 1100101101110010

Hamming

Emisor:

```
PS D:\2024\Semestre_2\Redes\lab2> & "C:/Users/Pablo Zamora/
o/hamming/hamming_encoder.py
Ingrese la cadena de bits a codificar: 1100101101110010
Datos codificados: 001110011011011010010
PS D:\2024\Semestre_2\Redes\lab2>
```

Mensaje codificado: 1100101101110010

Se introduce un error en la posición 12: 1100101101100010

```
PS D:\2024\Semestre_2\Redes\lab2\repo\hamming> node hamming_decoder.js
Ingrese la cadena de bits codificada: 001110011011011010010
Datos codificados con error introducido: 001110011010011010010
Bit incorrecto en la posición: 12. Corrigiendo...
Datos decodificados: 1100101101110010
PS D:\2024\Semestre_2\Redes\lab2\repo\hamming>
```

Fletcher16 checksum

Emisor:

```
PS D:\diego\OneDrive - UVG\Documentos Universidad\Semestre 8\r
Ingresar mensaje en binario a codificar: 1100101101110010
Mensaje completo codificado:
1100101101110010000101000111110
PS D:\diego\OneDrive - UVG\Documentos Universidad\Semestre 8\r
```

Mensaje codificado: 1100101101110010000101000111110

Receptor:

Se introduce un error en la posición 6: 110010**1**101110010000101000111110

```
PS D:\diego\OneDrive - UVG\Documentos Universidad\Semestre 8\rede
iego/OneDrive - UVG/Documentos Universidad/Semestre 8/redes/labs/
Ingresar cadena codificada: 1100100101110010000101000111110
Checksum incorrecto. El mensaje contiene errores.
PS D:\diego\OneDrive - UVG\Documentos Universidad\Semestre 8\rede
```

Dos o más errores

Cabe mencionar que el **código de Hamming** no es capaz de identificar más de un error en una cadena de bits, por lo que el bit en el que detecte error será incorrecto (a menos que por mera coincidencia acierte con uno de los errores).

- Cadena a codificar: 111101011

Hamming

Emisor:

```
PS D:\2024\Semestre_2\Redes\lab2> & "C:/Users/Pablo
/lab2/repo/hamming/hamming_encoder.py
Ingrese la cadena de bits a codificar: 111101011
Datos codificados: 0011111101011
PS D:\2024\Semestre_2\Redes\lab2>
```

Mensaje codificado: 0011111101011

Receptor:

Se introduce un error en las posiciones 1, 3 y 13: **1**0**0**1111110100**0**1

```
PS D:\2024\Semestre_2\Redes\lab2\repo\hamming> node hamming_decoder.js
Ingrese la cadena de bits codificada: 10011111101001
Bit incorrecto en la posición: 5. Corrigiendo...
Datos decodificados: 0011101001
PS D:\2024\Semestre_2\Redes\lab2\repo\hamming>
```

Fletcher16 checksum

Emisor:

```
PS D:\diego\OneDrive - UVG\Documentos Universidad\Semest
Ingresar mensaje en binario a codificar: 111101011
Mensaje completo codificado:
111101011111010111101100
PS D:\diego\OneDrive - UVG\Documentos Universidad\Semest
```

Mensaje codificado: 111101011111010111101100

Receptor:

Se introduce un error en las posiciones 1, 3 y 13: **1**11101011111**0**110111101100

```
PS D:\diego\OneDrive - UVG\Documentos Universidad\Semestre 8\redes\labs\
iego/OneDrive - UVG/Documentos Universidad/Semestre 8/redes/labs/lab2/fl
Ingresar cadena codificada: 010101011111111010111101100
Checksum incorrecto. El mensaje contiene errores, descartando...
PS D:\diego\OneDrive - UVG\Documentos Universidad\Semestre 8\redes\labs\
```

- Cadena a codificar: 10101010

Hamming

Emisor:

```
PS D:\2024\Semestre_2\Redes\lab2> & "C:/Users/Pablo
o/hamming/hamming_encoder.py
Ingrese la cadena de bits a codificar: 10101010
Datos codificados: 111101001010
PS D:\2024\Semestre_2\Redes\lab2>
```

Mensaje codificado: 111101001010

Receptor:

Se introduce error en las posiciones 2 y 4: 1**0**1**0**01001010

```
PS D:\2024\Semestre_2\Redes\lab2\repo\hamming> node hamming_decoder.js
Ingrese la cadena de bits codificada: 101001001010
Bit incorrecto en la posición: 6. Corrigiendo...
Datos decodificados: 10001010
PS D:\2024\Semestre_2\Redes\lab2\repo\hamming>
```

Fletcher16 checksum

Emisor:

```
PS D:\diego\OneDrive - UVG\Documentos Universidad\Semestre
Ingresar mensaje en binario a codificar: 10101010
Mensaje completo codificado:
1010101010101010101010
PS D:\diego\OneDrive - UVG\Documentos Universidad\Semestre
```

Mensaje codificado: 1010101010101010101010

Receptor:

Se introduce error en los bits 2 y 4: 1**0**1**0**1010101010101010101010

```
PS D:\diego\OneDrive - UVG\Documentos Universidad\Semestre 8\redes\
iego/OneDrive - UVG/Documentos Universidad/Semestre 8/redes/labs/lab
Ingresar cadena codificada: 1111101010101010101010
Checksum incorrecto. El mensaje contiene errores, descartando...
PS D:\diego\OneDrive - UVG\Documentos Universidad\Semestre 8\redes\
```

- Cadena a codificar: 111110110101

Hamming

Emisor:

```
PS D:\2024\Semestre_2\Redes\lab2> & "C:/Users/Pablo
o/hamming/hamming_encoder.py
Ingrese la cadena de bits a codificar: 111110110101
Datos codificados: 01111110101101011
PS D:\2024\Semestre_2\Redes\lab2>
```

Mensaje codificado: 01111110101101011

Receptor:

Se introduce error en las posiciones 5, 10, 12 y 15:

0111**0**1101**1**1**0**01**1**11

```
PS D:\2024\Semestre_2\Redes\lab2\repo\hamming> node hamming_decoder.js
Ingrese la cadena de bits codificada: 01110110111001111
Bit incorrecto en la posición: 12. Corrigiendo...
Datos decodificados: 101111110111
PS D:\2024\Semestre_2\Redes\lab2\repo\hamming>
```

<u>Fletcher16 checksum</u>

Emisor:

```
PS D:\diego\OneDrive - UVG\Documentos Universidad\Semest
Ingresar mensaje en binario a codificar: 111110110101
Mensaje completo codificado:
1111101101011101001111000100
PS D:\diego\OneDrive - UVG\Documentos Universidad\Semest
```

Cadena codificada: 1111101101011101001111000100

Receptor:

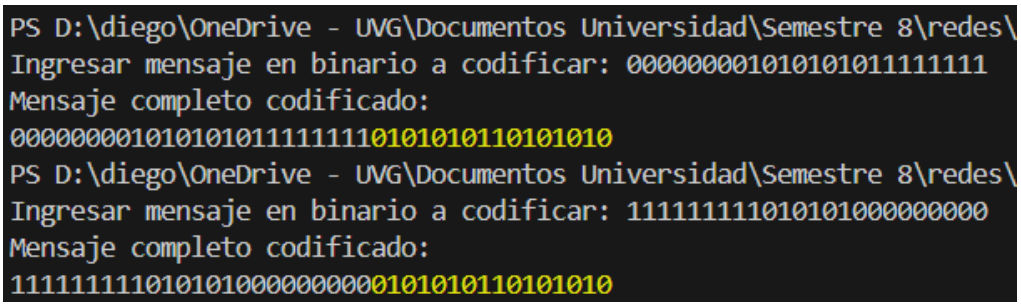
Se introduce error en las posiciones 5, 10, 12 y 15:
1111**1**0110**1**0**1**11**0**1001111000100

```
PS D:\diego\OneDrive - UVG\Documentos Universidad\Semestre 8\redes\l
iego/OneDrive - UVG/Documentos Universidad/Semestre 8/redes/labs/lab
Ingresar cadena codificada: 1111001100001111001111000100
Checksum incorrecto. El mensaje contiene errores, descartando...
PS D:\diego\OneDrive - UVG\Documentos Universidad\Semestre 8\redes\l
```

¿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, demuestrelo con su implementación.

Sí, en el caso del algoritmo de **Fletcher Checksum** es posible manipular la combinación de bits con el objetivo de que no sea capaz de detectar el error correctamente. Esto se da pues el algoritmo no es capaz de distinguir entre bloques (en el caso de Fletcher-16, bloques de 8 bits) conformado por todos los bits cero y todos los bits 1. Por lo que, si un bloque completo de datos cambia de ceros a unos, la suma de comprobación permanece igual.

Por ejemplo, en la cadena de bits 00000000 10101010 11111111, los bloques 1 y 3 pueden ser cambiados por los bloques 11111111 y 00000000 respectivamente, sin que el algoritmo sea capaz de detectar el error.



Al calcular el checksum de ambas cadenas de bits diferentes (resaltado en amarillo), se obtiene el mismo resultado, por lo que al realizar la verificación no se detecta error alguno al intercambiar su contenido.

Esto se debe principalmente al overflow que se genera cuando el acumulador sum1 o sum2 alcanza 255. Al sumar un un valor adicional, el resultado "desborda" y vuelve a comenzar desde 0 debido a la aritmética módulo 255. Al aplicar la operación módulo 255 tanto a un bloque equivalente a 0 o 255 decimal, se obtiene el mismo resultado.

Esto también se cumple en el caso del algoritmo de **Código de Hamming.** Como se mostró en los escenarios de prueba, simplemente con añadir más de un error en el mensaje codificado, el algoritmo ya no es capaz de identificarlos. Esto se debe a que, en la decodificación, el algoritmo utiliza los bits de paridad establecidos en la codificación para identificar cuál de los bits de data incumple con dicha paridad. Al introducir más de un bit de error, la paridad ya no es un criterio confiable para identificar errores, y el cálculo de la posición pierde sentido.

Por ejemplo al codificar el mensaje 1010 y cambiar los últimos dos bits del mensaje codificado:

```
PS D:\2024\Semestre_2\Redes\lab2> & "C:/Users/Pablo Zamora/AppData/Local/Programs/Python/Python310/python.exe"
/lab2/repo/hamming/hamming_encoder.py
Ingrese la cadena de bits a codificar: 1010
Datos codificados: 1011010
PS D:\2024\Semestre_2\Redes\lab2> cd repo/hamming
PS D:\2024\Semestre_2\Redes\lab2\repo\hamming> node hamming_decoder.js
Ingrese la cadena de bits codificada: 1011001
Bit incorrecto en la posición: 1. Corrigiendo...
Datos decodificados: 1001
PS D:\2024\Semestre_2\Redes\lab2\repo\hamming>
```

Como se observa, al introducir error en los últimos dos bits de la cadena codificada, el algoritmo ya no es capaz de identificarlos.

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.

En el algoritmo de **Código de Hamming** se presentan varias ventajas respecto a los demás: Primero, es posible calcular dinámicamente la cantidad de bits de paridad que posee o poseerá el mensaje codificado, gracias a la condición $m + r + 1 \leq 2^r$, lo que facilita bastante tanto la codificación como la decodificación. Segundo, el uso de la operación bit a bit AND (&) permite identificar fácilmente tanto si una posición de la cadena es una potencia de 2, como si una posición de la cadena influye en un bit de paridad. Finalmente, la operación bit a bit XOR (^) permite obtener fácilmente la paridad de una subcadena del mensaje. El hecho de que sea posible utilizar estas operaciones en el algoritmo (en lugar de potenciación, por ejemplo), hace que este sea bastante eficiente.

Sin embargo, como ya se mencionó anteriormente, este algoritmo presenta la gran desventaja de ser incapaz de reconocer más de un error en el mensaje, ya que los bits de paridad únicamente permiten identificar, y posteriormente corregir, la posición de un bit de error.

El algoritmo **Fletcher Checksum** presenta una complejidad baja, gracias a su simplicidad derivada del uso exclusivo de sumas de verificación básicas. Esto facilita su implementación y minimiza el consumo de recursos durante su ejecución. Como resultado, permite una verificación rápida, siendo ideal para reducir la latencia en la transmisión y recepción de paquetes. El Fletcher-16, en particular, tiene una redundancia baja debido a que el tamaño de la suma de verificación (16 bits) se mantiene constante sin importar la longitud de los datos a verificar. Además, puede detectar uno o más errores sin limitaciones específicas en este aspecto.

Sin embargo, una limitación importante es que Fletcher-16, a diferencia del código de Hamming, no está diseñado para la corrección de errores, sino solo para la detección. Además, puede ser susceptible a errores, especialmente en casos donde ocurren múltiples errores en el mismo paquete de datos.