Universidad del Valle de Guatemala
Redes CC3067
Kenneth Galvez 20079
Natanael Giron 20029

Laboratorio 2 - Primera parte
Esquemas de detección y corrección de errores

- Descripción de la práctica

En el Laboratorio 2 - Primera parte, se trabajó con la temática de esquemas de detección y corrección de errores en sistemas de comunicación. Tras este laboratorio, reconocemos que el ruido y los errores de transmisión son fenómenos comunes en cualquier tipo de comunicación y es importante aprender a manejar adecuadamente estas fallas para garantizar la integridad y la fiabilidad de la información transmitida.

Los objetivos de esta práctica son:

- Comprender en detalle el funcionamiento de los algoritmos de detección y corrección de errores, así como sus ventajas y desventajas.
- Implementar al menos un algoritmo de detección de errores y otro de corrección de errores.
- Implementar cada algoritmo en dos lenguajes de programación distintos, de forma que un receptor en un lenguaje pueda validar un mensaje codificado por el mismo algoritmo en otro lenguaje.

Nuestra pareja escogió el “Código de Hamming “ para corrección de errores y el “Fletcher Checksum” para detección de errores. El código de Hamming es un código detector y corrector de errores que lleva el nombre de su inventor, Richard Hamming. Este código se utiliza para detectar y corregir errores en la transmisión de datos. La idea detrás de los códigos de Hamming es intercalar, o agregar, dígitos binarios adicionales a un código binario para que puedan detectarse y corregirse errores en la transmisión del código a través de un canal (7.5: Códigos Hamming Para Codificación de Canales, 2022). La lógica dice que para cualquier código (n,m) que cumpla (m + r + 1) <= 2^r, se pueden agregar r bits de paridad para detectar y corregir errores.

El codigo de Fletcher Checksum se trata de una técnica de código de bloques ideada por John G. Fletcher en los años 70 en los laboratorios Lawrence Livermore (EE.UU.). Este código es un algoritmo de detección de errores que se utiliza para verificar la integridad de los datos transmitidos. El algoritmo divide los datos en bloques de 8, 16 o 32 bits y calcula dos sumas de verificación para cada bloque. Estas sumas se combinan para formar un valor de verificación de Fletcher checksum. (Chandu yadav, 2020)

Universidad del Valle de Guatemala
Redes CC3067
Kenneth Galvez 20079
Natanael Giron 20029

- Resultados

Código de Hamming (Emisor: Python | Receptor: JavaScript)

- Caso 1: Tramas sin errores
  - Trama 1 (Sin errores):
    Emisor: 1001101
    Receptor (al recibir): 01110010101
    Resultado esperado: El receptor mostrará "No se detectaron errores" y la trama original.

```
>>> %Run emisor.py
 Ingrese la trama de bits (ejemplo: 1010): 1001101
 Código de Hamming: 01110010101
```

  - Trama 2 (Sin errores):
    Emisor: 0011010
    Receptor (al recibir): 11000111010
    Resultado esperado: El receptor mostrará "No se detectaron errores" y la trama original.

```
>>> %Run emisor.py
 Ingrese la trama de bits (ejemplo: 1010): 0011010
 Código de Hamming: 11000111010
```

- Trama 3 (Sin errores):
  Emisor: 0110110
  Receptor (al recibir): 00001100110
  Resultado esperado: El receptor mostrará "No se detectaron errores" y la trama original.

```
>>> %Run emisor.py
  Ingrese la trama de bits (ejemplo: 1010): 0110110
  Código de Hamming: 00001100110
```

Receptor de Hamming

Ingrese el código de Hamming recibido: 00001100110 [Decodificar]
No se encontraron errores.
Datos recibidos: 0, 0, 0, 0, 1, 1, 0, 0, 1, 1, 0
Trama Original: 0, 1, 1, 0, 1, 1, 0

- Caso 2: Tramas con errores detectables
  - Trama 4 (Con error 1 bit):
    Emisor: 1110001
    Receptor (al recibir, con el último bit modificado): 11101101001
    Resultado esperado: El receptor mostrará "Se encontró un error en la posición 11. Datos recibidos con el error corregido."

```
>>> %Run emisor.py
  Ingrese la trama de bits (ejemplo: 1010): 1110001
  Código de Hamming: 11101101001
```

Receptor de Hamming

Ingrese el código de Hamming recibido: 11101101000 [Decodificar]
Se encontró un error en la posición: 11
Datos recibidos con el error corregido: 1, 1, 1, 0, 1, 1, 0, 1, 0, 0, 1

- Trama 5 (Con error 1 bit):
  Emisor: 01010011
  Receptor (al recibir, con el último bit modificado): 000110100011
  Resultado esperado: El receptor mostrará "Se encontró un error en la posición 1. Datos recibidos con el error corregido."

```
>>> %Run emisor.py
 Ingrese la trama de bits (ejemplo: 1010): 01010011
 Código de Hamming: 100110100011
```

Receptor de Hamming

Ingrese el código de Hamming recibido: 000110100011 Decodificar
Se encontró un error en la posición: 1
Datos recibidos con el error corregido: 1, 0, 0, 1, 1, 0, 1, 0, 0, 0, 1, 1

- Trama 6 (Con error 1 bit):
  Emisor: 1001000
  Receptor (al recibir, con el último bit modificado): 001101110000
  Resultado esperado: El receptor mostrará "Se encontró un error en la posición 6. Datos recibidos con el error corregido."

```
>>> %Run emisor.py
 Ingrese la trama de bits (ejemplo: 1010): 1001000
 Código de Hamming: 00110010000
```

Receptor de Hamming

Ingrese el código de Hamming recibido: 00110110000 Decodificar
Se encontró un error en la posición: 6
Datos recibidos con el error corregido: 0, 0, 1, 1, 0, 0, 1, 0, 0, 0, 0

Universidad del Valle de Guatemala
Redes CC3067
Kenneth Galvez 20079
Natanael Giron 20029

- Caso 3: Tramas con errores de 2 bit
  - Trama 7 (Con errores 2 bit):
    Emisor: 1110001
    Receptor (al recibir bits modificados): 11100101000
    Resultado esperado: El receptor mostrará "Se encontró un error", pero será en una posición incorrecta, ya que el código de hamming no puede corregir 2 errores

```
>>> %Run emisor.py
 Ingrese la trama de bits (ejemplo: 1010): 1110001
 Código de Hamming: 11101101001
```

Receptor de Hamming

Ingrese el código de Hamming recibido: 11100101000 Decodificar
Se encontró un error en la posición: 14
Datos recibidos con el error corregido: 1, 1, 1, 0, 0, 1, 0, 1, 0, 0, 0

  - Trama 8 (Con errores 2 bit):
    Emisor: 0111100
    Receptor (al recibir bits modificados): 10010011100
    Resultado esperado: El receptor mostrará "Se encontró un error", pero será en una posición incorrecta, ya que el código de hamming no puede corregir 2 errores

```
>>> %Run emisor.py
 Ingrese la trama de bits (ejemplo: 1010): 0111100
 Código de Hamming: 10011111100
```

Receptor de Hamming

Ingrese el código de Hamming recibido: 10010011100 Decodificar
Se encontró un error en la posición: 3
Datos recibidos con el error corregido: 1, 0, 1, 1, 0, 0, 1, 1, 1, 0, 0

- Trama 9 (Con errores 2 bit):
  Emisor: 1111100
  Receptor (al recibir bits modificados): 0111111100
  Resultado esperado: El receptor mostrará "Se encontró un error", pero será en una posición incorrecta, ya que el código de hamming no puede corregir 2 errores

```
>>> %Run emisor.py
 Ingrese la trama de bits (ejemplo: 1010): 1111100
 Código de Hamming: 01111111100
```

Receptor de Hamming

Ingrese el código de Hamming recibido: 01111111111 Decodificar
Se encontró un error en la posición: 1
Datos recibidos con el error corregido: 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1

- Caso 4: Trama no detectable por el receptor
  - Trama 10 (No detectable por el receptor):
    Emisor: 0000000
    Receptor : 00000000000
    Resultado esperado: El receptor mostrará "No se detectaron errores. Trama recibida: 0000000".

```
>>> %Run emisor.py
 Ingrese la trama de bits (ejemplo: 1010): 0000000
 Código de Hamming: 00000000000
```

Receptor de Hamming

Ingrese el código de Hamming recibido: 0000000000 Decodificar
No se encontraron errores.
Datos recibidos: 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
Trama Original: 0, 0, 0, 0, 0, 0

Universidad del Valle de Guatemala
Redes CC3067
Kenneth Galvez 20079
Natanael Giron 20029

Fletcher Checksum (Emisor: Python | Receptor: JavaScript)

- Caso 1: Tramas sin errores
  - Trama 1 (Sin errores):
    Emisor: 110101
    Receptor (al recibir): 1101011101010011010100
    Resultado esperado: El receptor mostrará "No se detectaron errores. Trama recibida: 110101".

```
PS D:\Universidad\Octavo Semestre\Redes\Lab2.1 Fletcher> & C:/Users/kenic/AppData/Local/Programs/Python/Python311/python.exe "d:/Universidad/Octavo Se
mestre/Redes/Lab2.1 Fletcher/fletcher.py"
Ingresa la trama en binario (e.g., '110101'): 110101
Ingresa el tamaño del bloque (8, 16 o 32): 8
Trama enviada: 1101011101010011010100
PS D:\Universidad\Octavo Semestre\Redes\Lab2.1 Fletcher> node index
Ingresa la cadena mostrada por el emisor: 1101011101010011010100
Ingresa el tamaño del bloque (8, 16 o 32): 8
No se detectaron errores. Trama recibida: 110101
```

  - Trama 2 (Sin errores):
    Emisor: 101011
    Receptor (al recibir): 1010111010110010101100
    Resultado esperado: El receptor mostrará "No se detectaron errores. Trama recibida: 101011".

```
PS D:\Universidad\Octavo Semestre\Redes\Lab2.1 Fletcher> & C:/Users/kenic/AppData/Local/Programs/Python/Python311/python.exe "d:/Universidad/Octavo Se
mestre/Redes/Lab2.1 Fletcher/fletcher.py"
Ingresa la trama en binario (e.g., '110101'): 101011
Ingresa el tamaño del bloque (8, 16 o 32): 8
Trama enviada: 1010111010110010101100
PS D:\Universidad\Octavo Semestre\Redes\Lab2.1 Fletcher> node index
Ingresa la cadena mostrada por el emisor: 1010111010110010101100
Ingresa el tamaño del bloque (8, 16 o 32): 8
No se detectaron errores. Trama recibida: 101011
```

  - Trama 3 (Sin errores):
    Emisor: 001110
    Receptor (al recibir): 0011100010011100
    Resultado esperado: El receptor mostrará "No se detectaron errores. Trama recibida: 001110".

```
PS D:\Universidad\Octavo Semestre\Redes\Lab2.1 Fletcher> & C:/Users/kenic/AppData/Local/Programs/Python/Python311/python.exe "d:/Universidad/Octavo Se
mestre/Redes/Lab2.1 Fletcher/fletcher.py"
Ingresa la trama en binario (e.g., '110101'): 001110
Ingresa el tamaño del bloque (8, 16 o 32): 16
Trama enviada: 0011100011100000111000
PS D:\Universidad\Octavo Semestre\Redes\Lab2.1 Fletcher> node index
Ingresa la cadena mostrada por el emisor: 0011100011100000111000
Ingresa el tamaño del bloque (8, 16 o 32): 16
No se detectaron errores. Trama recibida: 001110
```

- Caso 2: Tramas con errores detectables
  - Trama 4 (Con error 1 bit):
    Emisor: 011000
    Receptor (al recibir, con el último bit modificado): 011000011000000110000**1**
    Resultado esperado: El receptor mostrará "Se detectaron errores. La trama se descarta por detectar errores."

```
PS D:\Universidad\Octavo Semestre\Redes\Lab2.1 Fletcher> & C:/Users/kenic/AppData/Local/Programs/Python/Python311/python.exe "d:/Universidad/Octavo Se
mestre/Redes/Lab2.1 Fletcher/fletcher.py"
Ingresa la trama en binario (e.g., '110101'): 011000
Ingresa el tamaño del bloque (8, 16 o 32): 8
Trama enviada: 0110000110000001100000
PS D:\Universidad\Octavo Semestre\Redes\Lab2.1 Fletcher> node index
Ingresa la cadena mostrada por el emisor: 0110000110000001100001
Ingresa el tamaño del bloque (8, 16 o 32): 8
Se detectaron errores. La trama se descarta por detectar errores.
```

  - Trama 5 (Con error 1 bit):
    Emisor: 111100
    Receptor (al recibir, con el primer bit modificado): **0**111001111000011110000
    Resultado esperado: El receptor mostrará "Se detectaron errores. La trama se descarta por detectar errores."

```
PS D:\Universidad\Octavo Semestre\Redes\Lab2.1 Fletcher> & C:/Users/kenic/AppData/Local/Programs/Python/Python311/python.exe "d:/Universidad/Octavo Se
mestre/Redes/Lab2.1 Fletcher/fletcher.py"
Ingresa la trama en binario (e.g., '110101'): 111100
Ingresa el tamaño del bloque (8, 16 o 32): 32
Trama enviada: 11110011110000111100000
PS D:\Universidad\Octavo Semestre\Redes\Lab2.1 Fletcher> node index
Ingresa la cadena mostrada por el emisor: 01110011110000111100000
Ingresa el tamaño del bloque (8, 16 o 32): 32
Se detectaron errores. La trama se descarta por detectar errores.
```

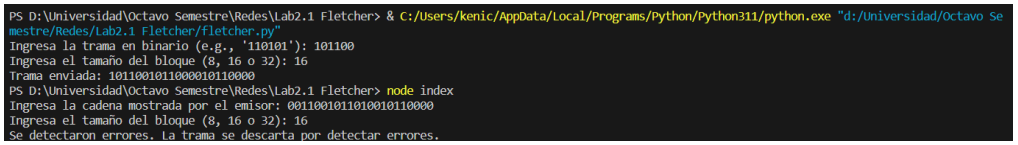
  - Trama 6 (Con error 1 bit):
    Emisor: 100101
    Receptor (al recibir, con un bit intermedio modificado):
    1001011001**1**10010010100
    Resultado esperado: El receptor mostrará "Se detectaron errores. La trama se descarta por detectar errores."

```
PS D:\Universidad\Octavo Semestre\Redes\Lab2.1 Fletcher> & C:/Users/kenic/AppData/Local/Programs/Python/Python311/python.exe "d:/Universidad/Octavo Se
mestre/Redes/Lab2.1 Fletcher/fletcher.py"
Ingresa la trama en binario (e.g., '110101'): 100101
Ingresa el tamaño del bloque (8, 16 o 32): 16
Trama enviada: 1001011001010010010100
PS D:\Universidad\Octavo Semestre\Redes\Lab2.1 Fletcher> node index
Ingresa la cadena mostrada por el emisor: 1001011001110010010100
Ingresa el tamaño del bloque (8, 16 o 32): 16
Se detectaron errores. La trama se descarta por detectar errores.
```

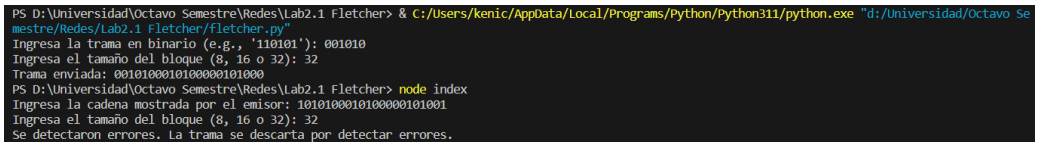
- Caso 3: Tramas con errores de 2 bit
  - Trama 7 (Con errores 2 bit):
    Emisor: 010011
    Receptor (al recibir, con el último bit y otro bit interm
    edio modificados): 0100110101**1**110001001101**1**
    Resultado esperado: El receptor mostrará "Se detectaron errores. La trama se descarta por detectar errores."

```
PS D:\Universidad\Octavo Semestre\Redes\Lab2.1 Fletcher> & C:/Users/kenic/AppData/Local/Programs/Python/Python311/python.exe "d:/Universidad/Octavo Se
mestre/Redes/Lab2.1 Fletcher/fletcher.py"
Ingresa la trama en binario (e.g., '110101'): 010011
Ingresa el tamaño del bloque (8, 16 o 32): 8
Trama enviada: 0100110100110001001100
PS D:\Universidad\Octavo Semestre\Redes\Lab2.1 Fletcher> node index
Ingresa la cadena mostrada por el emisor: 0100110101110001001101
Ingresa el tamaño del bloque (8, 16 o 32): 8
Se detectaron errores. La trama se descarta por detectar errores.
```

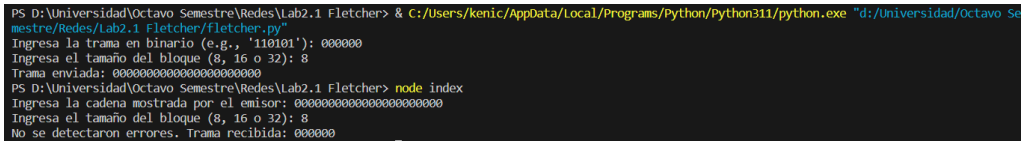
- Trama 8 (Con errores 2 bit):
  Emisor: 101100
  Receptor (al recibir, con el primer bit y otro bit intermedio modificados):
  001100101101001011000 0
  Resultado esperado: El receptor mostrará "Se detectaron errores. La trama se descarta por detectar errores."



- Trama 9 (Con errores 2 bit):
  Emisor: 001010
  Receptor (al recibir, con el último bit y el primer bit modificados):
  1010100010100000101001
  Resultado esperado: El receptor mostrará "Se detectaron errores. La trama se descarta por detectar errores."



- Caso 4: Trama no detectable por el receptor
  - Trama 10 (No detectable por el receptor):
    Emisor: 000000
    Receptor : 00000000000000000000
    Resultado esperado: El receptor mostrará "No se detectaron errores. Trama recibida: 000000".



- Discusión

El código Hamming tiene una fuerte capacidad de detección y puede averiguar si hay un error en los datos transmitidos. Sin embargo, su capacidad de corrección es limitada, especialmente cuando se enfrentan a errores de más de un bit. Esto se debe a que el código de Hamming se basa en la introducción de bits de paridad, y

cuando ocurren múltiples errores, la información redundante es insuficiente para determinar la ubicación exacta del error.

Fue interesante observar cómo el código Hamming se desempeñó satisfactoriamente durante las pruebas, mostrando su capacidad para detectar y corregir errores de un solo bit de manera eficiente. Sin embargo, el algoritmo muestra sus limitaciones cuando se trata de errores de 2 bits.

Hablando ahora del algoritmo de Fletcher Checksum utilizado para la detección de errores, pudimos observar varias ventajas y desventajas de este algoritmo. Su mayor fortaleza fue que no tuvo ningún inconveniente al momento de verificar algún error, siendo acertado en todas nuestras pruebas (esto debido a su manera de identificar cambios en los bits de cada trama). Otra fortaleza fue que resultó sencillo de implementar en ambos lenguajes. Su mayor fortaleza es su mayor limitación pues solo es capaz de identificar que ocurrió un error, pero no puede recuperar la información original. Resaltar también que para las pruebas de Fletcher se utilizaron tramas de seis bits debido a que considero que esta cantidad era lo suficientemente pequeña como para transmitir datos rápidamente, lo que resultó beneficioso para la cantidad de pruebas que se llevó a cabo.

Pensamos en trabajar en los lenguajes de programación Python (emisor) y Javascript (Receptor) ya que estos lenguajes han demostrado una alta eficiencia al momento de trabajar con ellos, y son los lenguajes de programación que encabezan los puestos de trabajo de la comunidad tecnológica, por lo que pensamos que sería útil utilizar lenguajes de alto rango para laboratorios como estos, tanto por su disponibilidad de uso, como la experiencia de los mismos trabajando de manera eficiente.

- Comentario grupal sobre el tema (errores)

Ambos algoritmos exhiben diferentes fortalezas y limitaciones en términos de detección y corrección de errores. Los códigos de Hamming son conocidos por su capacidad para corregir errores de un solo bit, lo que los hace útiles en escenarios donde la corrección precisa es fundamental. Sin embargo, en presencia de un error de dos bits, su rendimiento se degrada sustancialmente, por lo que se requiere una cuidadosa consideración al seleccionar un algoritmo para una aplicación en particular.

Por otro lado, se ha demostrado que el algoritmo de suma de comprobación de Fletcher es robusto en la detección de errores, incluso para datos de longitud variable y errores de varios bits. Esto lo convierte en una opción atractiva para aplicaciones donde la detección confiable de errores es una prioridad y no se requiere una corrección precisa.

- Conclusiones

La selección del algoritmo de detección y corrección de errores depende de las necesidades y características específicas de la aplicación. El algoritmo de Hamming hace un gran trabajo al corregir errores de un solo bit, pero tiene problemas con múltiples errores.
El algoritmo de Fletcher Checksum, por otro lado, es más eficiente en la detección de errores en datos de longitud variable y en situaciones de múltiples errores.

La combinación de ambos algoritmos podría ofrecer una solución más completa y robusta para asegurar la integridad de los datos en escenarios que requieren tanto la corrección precisa como la detección confiable de errores.

Es fundamental realizar pruebas exhaustivas y considerar las características particulares de cada algoritmo antes de tomar una decisión, ya que esto asegurará una detección y corrección de errores óptima para la aplicación en cuestión.

- Citas y Referencias

- 7.5: Códigos Hamming para codificación de canales. (2022, October 30). LibreTexts Español. https://espanol.libretexts.org/Ingenieria/Un_Primer_Curso_de_Ingenier%C3%ADa_El%C3%A9ctrica_e_Inform%C3%A1tica_(Scharf)/07%3A_C%C3%B3digos_binarios/7.05%3A_C%C3%B3digos_Hamming_para_codificaci%C3%B3n_de_canales
- Chandu yadav. (2020). Tutorialspoint.com. https://www.tutorialspoint.com/fletcher-s-checksum