Universidad del Valle de Guatemala
Redes

Lourdes Saavedra 21333
Gabriela Archila 21037

LABORATORIO 2 – PRIMERA PARTE

ESQUEMAS DE DETECCIÓN Y CORRECCIÓN DE ERRORES

Descripción de la práctica

En esta práctica se implementaron dos algoritmos para la detección y corrección de errores en la transmisión de datos: el Código de Hamming para la corrección de errores y el Fletcher Checksum para la detección de errores. El emisor de cada algoritmo fue desarrollado en Java, mientras que el receptor se implementó en Python, con el objetivo de cumplir el requisito usar lenguajes distintos para cada componente. El objetivo principal fue analizar el funcionamiento de estos algoritmos, evaluar su eficacia en la identificación y corrección de errores y comparar sus ventajas y desventajas en términos de complejidad, redundancia y capacidad de manejo de errores. Se realizaron pruebas con tramas de diferentes longitudes, simulando casos sin errores, con un error y con múltiples errores, para verificar la robustez de cada algoritmo. Además, se exploraron las limitaciones de cada método como la posibilidad de que errores específicos no sean detectados.

Tramas utilizadas y resultados:

A continuación se presentan las tramas utilizadas para ambos algoritmos en los diferentes casos. Con el objetivo de tener una base y poder comparar dichos algoritmos se utilizaron los mismos mensajes. Así pues, los mensajes enviados a los emisores fueron los siguientes: **0101**, **11000011**, **1010101010**. El primero tiene 4 bits, el segundo tiene 8 bits, y el tercero tiene 10 bits.

Luego de esto se procede a copiar el mensaje generado por el emisor y proporcionarlo al receptor en tres diferentes situaciones: tal cual como fue generado, cambiando un bit cualquiera y cambiando dos o más bits cualesquiera. Los mensajes utilizados se muestran a continuación:

Algoritmo Hamming:

- Caso sin errores:
  Receptor: 0100101, 101010000011, 101101001010101

- Caso con un error:
  Receptor: 0100001, 111010000011, 101101101010101

- Caso con dos errores:

Receptor: 0110111, 101100000011, 100101001010111

Algoritmo Fletcher checksum

- Caso sin errores:
  Receptor: 010100000101000001010000, 110000111100001111000011, 101010101010000000100101111110101

- Caso con un error:
  Receptor: 010100000110100000101000, 110000011100001111000011, 101010101010010001001001011111110101

- Caso con dos errores:
  Receptor: 010101000010100100101000, 110010111100001110000011, 101000101010000000100101110110101

Tramas devueltas

Ahora, las tramas devueltas por los algoritmos fueron las siguientes:

Algoritmo Hamming

- Emisor :
  Outputs: 0100101, 101010000011, 101101001010101

```
Ingrese la trama: 0101
Algoritmo de Hamming:  r1 = 0
r2 = 1
r4 = 0
Trama con c?digo de Hamming: 0100101
```

```
Ingrese la trama: 11000011
Algoritmo de Hamming:  r1 = 1
r2 = 0
r4 = 0
r8 = 0
Trama con c?digo de Hamming: 101010000011
```

```
Ingrese la trama: 10101010101
Algoritmo de Hamming:  r1 = 1
r2 = 0
r4 = 1
r8 = 0
Trama con c?digo de Hamming: 101101001010101
```

- Caso sin errores:

```
Ingrese la trama codificada: 0100101
No se detectaron errores
Trama recibida: 0100101

Datos originales: 0101
```

```
Ingrese la trama codificada: 101010000011
No se detectaron errores
Trama recibida: 101010000011

Datos originales: 11000011
```

```
Ingrese la trama codificada: 101101001010101
No se detectaron errores
Trama recibida: 101101001010101

Datos originales: 10101010101
```

- Caso con un error:

```
Ingrese la trama codificada: 0100001
Error detectado en la posición 5
Trama recibida:  0100001
Trama corregida: 0100101

Datos originales: 0101
```

```
Ingrese la trama codificada: 111010000011
Error detectado en la posición 2
Trama recibida:  111010000011
Trama corregida: 101010000011

Datos originales: 11000011
```

```
Ingrese la trama codificada: 101101101010101
Error detectado en la posición 7
Trama recibida:  101101101010101
Trama corregida: 101101001010101

Datos originales: 10101010101
```

- Caso con dos errores:

```
Ingrese la trama codificada: 0110111
Error detectado en la posición 5
Trama recibida:  0110111
Trama corregida: 0110011

Datos originales: 1011
```

```
Ingrese la trama codificada: 101100000011
Error detectado en la posición 1
Trama recibida:  101100000011
Trama corregida: 001100000011

Datos originales: 10000011
```

```
Ingrese la trama codificada: 100101001010111
Error detectado en la posición 13
Trama recibida:  100101001010111
Trama corregida: 100101001010011

Datos originales: 00101010011
```

Algoritmo Fletcher checksum

- Emisor:
  Outputs: 010100000101000001010000, 110000111100001111000011, 101010101010000001001010111110101

```
Ingrese la trama en binario (ej. 11010110):
0101

Resultados:
Trama original: 01010000
Checksum (sum1): 80 -> 01010000
Checksum (sum2): 80 -> 01010000
Mensaje completo a enviar: 010100000101000001010000
```

```
Ingrese la trama en binario (ej. 11010110):
11000011

Resultados:
Trama original: 11000011
Checksum (sum1): 195 -> 11000011
Checksum (sum2): 195 -> 11000011
Mensaje completo a enviar: 110000111100001111000011
```

```
Ingrese la trama en binario (ej. 11010110):
10101010101

Resultados:
Trama original: 1010101010100000
Checksum (sum1): 75 -> 01001011
Checksum (sum2): 245 -> 11110101
Mensaje completo a enviar: 10101010101000000100101111110101
```

- Caso sin errores:

```
Ingrese el mensaje (trama + checksum de 16 bits):
01010000010100000 1010000
No se detectaron errores
Trama original: 01010000
```

```
Ingrese el mensaje (trama + checksum de 16 bits):
110000111100001111000011
No se detectaron errores
Trama original: 11000011
```

```
Ingrese el mensaje (trama + checksum de 16 bits):
1010101010100000010010111110101
No se detectaron errores
Trama original: 1010101010100000
```

- Caso con un error:

```
Ingrese el mensaje (trama + checksum de 16 bits):
01010000110100000 1010000
Se detectaron errores - Mensaje descartado
```

```
Ingrese el mensaje (trama + checksum de 16 bits):
110000011100001111000011
Se detectaron errores - Mensaje descartado
```

```
Ingrese el mensaje (trama + checksum de 16 bits):
10101010101001000100101111110101
Se detectaron errores - Mensaje descartado
```

- Caso con dos errores:

```
Ingrese el mensaje (trama + checksum de 16 bits):
010101000101001001010000
Se detectaron errores - Mensaje descartado
```

```
Ingrese el mensaje (trama + checksum de 16 bits):
110010111100001110000011
Se detectaron errores - Mensaje descartado
```

```
Ingrese el mensaje (trama + checksum de 16 bits):
101000101010000001001011110110101
Se detectaron errores - Mensaje descartado
```

Discusión

- ¿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

  Hamming

  El algoritmo Hamming puede tener casos en que es posible manipular los bits y que no detecte errores correctamente. Por ejemplo, para la trama 0101 devuelve 0100101, y al cambiarlo a 0110000, el algoritmo detecta un error en la posición 1. Sin embargo, esta es correcta según la trama que devolvió el emisor. La razón es que el algoritmo Hamming falla en detectar múltiples errores debido a que los cálculos de paridad se cancelan entre sí.

```
Ingrese la trama codificada: 0110000
Error detectado en la posición 1
Trama recibida:  0110000
Trama corregida: 1110000

Datos originales: 1000
```

  Fletcher checksum

  Es posible manipular los bits de forma que el Fletcher checksum no detecte los errores. Esto ocurre cuando los cambios en los bits se compensan exactamente

en ambas sumas (sum1 y sum2). Por ejemplo, si se modifican dos bits de tal manera que uno aumenta un valor en sum1 y el otro lo disminuye en la misma cantidad, manteniendo sum2 sin cambios, el checksum final seguirá coincidiendo. Sin embargo, estas situaciones son poco probables en transmisiones reales, especialmente con tramas largas. Es necesario tomar en cuenta que el algoritmo depende de sumas acumulativas modulares. Si los errores alteran los bloques de manera que la suma total de los cambios en sum1 ≡ 0 mod 255 y la suma acumulada de esos cambios en sum2 también ≡ 0 mod 255 el checksum no detectará el error.

- 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.
  Ejemplo: “En la implementación del bit de paridad par, me di cuenta que comparado con otros métodos, la redundancia es la mínima (1 bit extra). Otra ventaja es la facilidad de implementación y la velocidad de ejecución, ya que se puede obtener la paridad aplicando un XOR entre todos los bits. Durante las pruebas, en algunos casos el algoritmo no era capaz de detectar el error, esto es una desventaja, por ejemplo […].”
  <u>Hamming</u>
  La ventaja principal del algoritmo Hamming es que no solo detecta errores, sino que también puede corregirlos. Esto se logra al identificar la posición exacta del bit con error, lo que elimina la necesidad de una retransmisión de datos. Por esta razón, es muy eficaz en entornos con una baja frecuencia de errores. Además, su overhead es bajo, ya que solo requiere $\log_2(n)$ bits adicionales para tramas de n bits.

  No obstante, el algoritmo Hamming también tiene limitaciones. Solo puede corregir un único error por trama. Si una trama contiene más de un error, el algoritmo no podrá corregirlos, o incluso podría cambiar un bit que ya era correcto. Además, es importante mencionar que Hamming puede fallar en la detección cuando los cambios en los bits no son completamente aleatorios, ya que está diseñado para errores esporádicos. Adicionalmente, comparado con algoritmos como Fletcher checksum, Hamming es más lento debido a la complejidad de sus cálculos.

  <u>Fletcher checksum</u>
  Por un lado, el Fletcher checksum es fácil de implementar y rápido de ejecutar, ya que solo requiere sumas acumulativas y operaciones módulo. Además, detecta una amplia gama de errores, incluyendo errores por transposición (bits invertidos en posiciones distantes) y errores múltiples, como se vio en las

pruebas con tramas alteradas. Asimismo, su overhead es constante (16 bits para bloques de 8 bits o 32 bits para bloques de 16 bits), lo que lo hace eficiente para tramas largas. Además, al no depender de estructuras complejas como Hamming, es menos propenso a errores de implementación.

Por otro lado, la principal desventaja es que solo detecta errores, no los corrige. Esto obliga a retransmitir los datos en caso de fallos. Aunque es robusto, existen casos extremadamente raros donde cambios específicos en los bits pueden compensarse en las sumas *sum1* y *sum2*, haciendo que el error pase desapercibido (esto se demostró modificando bits estratégicamente en las pruebas). Además, para tramas muy cortas, el overhead puede ser proporcionalmente alto (ej.: añadir 16 bits a una trama de 8 bits).

Conclusiones

- os algoritmos Hamming y Fletcher Checksum presentan fallas frente a múltiples errores, ya que en algunos casos la manipulación de los bits puede cancelar entre sí los cálculos de paridad, o bien los cambios en los bits pueden compensarse en las sumas internas, respectivamente.
- El algoritmo Hamming es más útil para entornos con baja frecuencia de errores aleatorios en la detección.
- El algoritmo Fletcher checksum es más útil para una detección más robusta.