

Laboratorio 2 Parte 1

Esquemas de detección y corrección de errores

Para el laboratorio se trabajaron 2 algoritmos para la detección y corrección de errores, donde dada una trama aleatoria, un emisor, escrito en un lenguaje de programación, le envía a un receptor, escrito en otro lenguaje de programación, la trama codificada, teniendo este la tarea de decodificar el mensaje dado por el remitente.

Se trabajaron dos algoritmos: el CRC-32 y el Hamming.

CRC-32:

1. Para cualquier trama de longitud n , $M_n(x)$, y el polinomio estándar para CRC-32 (uno de 32 bits, investigar cual es), donde $n > 32$
2. Para la implementación de este laboratorio se trabajó con un polinomio de grado 3 para el dividendo, de la siguiente manera: 1101. Esto facilitó crear un algoritmo más compacto que simulará el funcionamiento del CRC-32 a una menor escala sin usar polinomios demasiado grandes como dividendos.
3. Sumado a esto solo se usaron 3 bits binarios de paridad en comparación a todos los que usan normalmente el codificador que hasta suele incluir letras.

Hamming:

1. A partir de los datos binarios originales, se añaden bits de paridad en posiciones específicas (posiciones de potencias de 2) para generar el código de Hamming. Estos bits de paridad se calculan de tal manera que la paridad de ciertos grupos de bits, que incluyen el bit de paridad y algunos bits de datos, sea par o impar.
2. Una vez generado el código de Hamming, este se transmite o almacena. Si se produce un error en un solo bit durante la transmisión o el almacenamiento, el código de Hamming puede detectarlo.
3. Al recibir el código de Hamming, se verifica la paridad de los mismos grupos de bits que se utilizaron para calcular los bits de paridad. Si la paridad verificada no coincide con el bit de paridad recibido, se identifica un error. La posición del error se determina a partir de los bits de paridad que contienen errores. El bit erróneo se corrige y se obtiene la data original.

Resultados

o Incluir las tramas utilizadas, las tramas devueltas por el emisor, indicar los bits cambiados de forma manual, y los mensajes del receptor para cada uno de los casos solicitados.

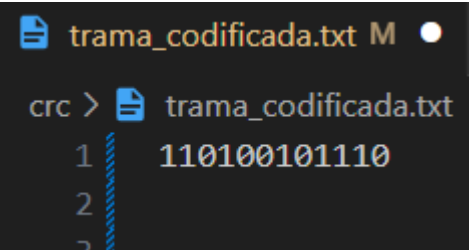
o Evidenciar sus pruebas con capturas de pantalla, etc..

CRC-32:

Tramas Correctas:

1.

```
PS C:\Users\Jose\Documents\GitHub\Fork\Lab1_Red<sup>es</sup>> python .\crc\emisor.py
Ingresa una trama ej: 11010011101100
>> 110100101
Mensaje original + CRC: 110100101110
El resultado se ha guardado en el archivo trama_codificada.txt
```

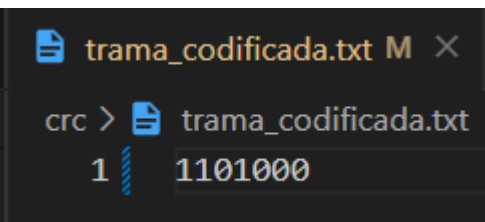


```
trama_codificada.txt M
crc > trama_codificada.txt
1 110100101110
2
3
```

```
PS C:\Users\Jose\Documents\GitHub\Fork\Lab1_Red<sup>es</sup>> node .\crc\receptor.js
Trama recibida: 110100101
```

2.

```
PS C:\Users\Jose\Documents\GitHub\Fork\Lab1_Red<sup>es</sup>> python .\crc\emisor.py
Ingresa una trama ej: 11010011101100
>> 1101
Mensaje original + CRC: 1101000
El resultado se ha guardado en el archivo trama_codificada.txt
```



```
trama_codificada.txt M X
crc > trama_codificada.txt
1 1101000
```

```
PS C:\Users\Jose\Documents\GitHub\Fork\Lab1_Red<sup>es</sup>> node .\crc\receptor.js
Trama recibida: 1101
```

3.

```
PS C:\Users\Jose\Documents\GitHub\Fork\Lab1_Red<sup>es</sup>> python .\crc\emisor.py
Ingresa una trama ej: 11010011101100
>> 11110000
Mensaje original + CRC: 11110000010
El resultado se ha guardado en el archivo trama_codificada.txt
```

```
trama_codificada.txt M X
crc > trama_codificada.txt
1 11110000010
```

Tramas Con 1 Error:

1.

```
PS C:\Users\Jose\Documents\GitHub\Fork\Lab1_Redes> python .\crc\emisor.py
Ingresa una trama ej: 11010011101100
>> 110100101
Mensaje original + CRC: 110100101110
El resultado se ha guardado en el archivo trama_codificada.txt
```

```
trama_codificada.txt M X
crc > trama_codificada.txt
1 110100101111
2
3
```

```
PS C:\Users\Jose\Documents\GitHub\Fork\Lab1_Redes> node .\crc\receptor.js
Trama descartada por detectar errores.
```

2.

```
PS C:\Users\Jose\Documents\GitHub\Fork\Lab1_Redes> python .\crc\emisor.py
Ingresa una trama ej: 11010011101100
>> 1101
Mensaje original + CRC: 1101000
El resultado se ha guardado en el archivo trama_codificada.txt
```

```
trama_codificada.txt M X
crc > trama_codificada.txt
1 1101010
```

```
PS C:\Users\Jose\Documents\GitHub\Fork\Lab1_Redes> node .\crc\receptor.js
Trama descartada por detectar errores.
```

3.

Redes

Laboratorio 2

Juan Angel Carrera 20593

José Mariano Reyes 20074

```
PS C:\Users\Jose\Documents\GitHub\Fork\Lab1_Redés> python .\crc\emisor.py
Ingresa una trama ej: 11010011101100
>> 11110000
Mensaje original + CRC: 11110000010
El resultado se ha guardado en el archivo trama_codificada.txt
```

```
trama_codificada.txt M X
crc > trama_codificada.txt
1 11110000011
```

```
PS C:\Users\Jose\Documents\GitHub\Fork\Lab1_Redés> node .\crc\receptor.js
Trama descartada por detectar errores.
```

Tramas con 2 Errores:

```
PS C:\Users\Jose\Documents\GitHub\Fork\Lab1_Redés> python .\crc\emisor.py
Ingresa una trama ej: 11010011101100
>> 110100101
Mensaje original + CRC: 110100101110
El resultado se ha guardado en el archivo trama_codificada.txt
```

```
trama_codificada.txt M X
crc > trama_codificada.txt
1 110100101100
2
```

```
PS C:\Users\Jose\Documents\GitHub\Fork\Lab1_Redés> node .\crc\receptor.js
Trama descartada por detectar errores.
```

2.

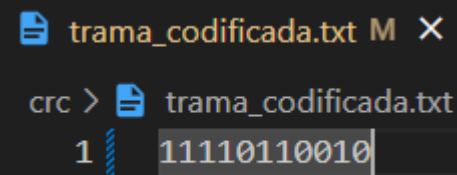
```
PS C:\Users\Jose\Documents\GitHub\Fork\Lab1_Redés> python .\crc\emisor.py
Ingresa una trama ej: 11010011101100
>> 1101
Mensaje original + CRC: 1101000
El resultado se ha guardado en el archivo trama_codificada.txt
```

```
trama_codificada.txt M X
crc > trama_codificada.txt
1 1101011
```

```
PS C:\Users\Jose\Documents\GitHub\Fork\Lab1_Redés> node .\crc\receptor.js  
Trama descartada por detectar errores.
```

3.

```
PS C:\Users\Jose\Documents\GitHub\Fork\Lab1_Redés> python .\crc\emisor.py  
Ingresa una trama ej: 11010011101100  
>> 11110000  
Mensaje original + CRC: 11110000010  
El resultado se ha guardado en el archivo trama_codificada.txt
```



trama_codificada.txt M X

crc > trama_codificada.txt

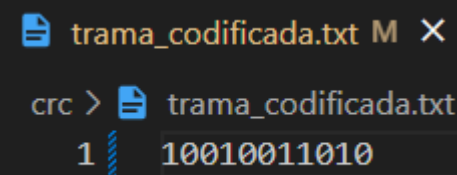
1 11110110010

```
PS C:\Users\Jose\Documents\GitHub\Fork\Lab1_Redés> node .\crc\receptor.js  
Trama descartada por detectar errores.
```

Tramas Modificadas pero funcionales:

1.

```
PS C:\Users\Jose\Documents\GitHub\Fork\Lab1_Redés> python .\crc\emisor.py  
Ingresa una trama ej: 11010011101100  
>> 110100101  
Mensaje original + CRC: 110100101110  
El resultado se ha guardado en el archivo trama_codificada.txt
```



trama_codificada.txt M X

crc > trama_codificada.txt

1 10010011010

```
PS C:\Users\Jose\Documents\GitHub\Fork\Lab1_Redés> node .\crc\receptor.js  
Trama corregida: 10010011110
```

Hamming:
Tramas Correctas:

1.

```
> python .\emisor.py
Introduzca una trama en binario:
1001
Mensaje codificado: 0011001
> node .\receptor.js
DATA: 0011001 Paridad: 3
No error is found
● Original data bits: 1001
```

2.

```
> python .\emisor.py
Introduzca una trama en binario:
1010
Mensaje codificado: 1011010
> node .\receptor.js
DATA: 1011010 Paridad: 3
No error is found
● Original data bits: 1010
```

3.

```
● > python .\emisor.py
Introduzca una trama en binario:
0011
Mensaje codificado: 1000011
> node .\receptor.js
DATA: 1000011 Paridad: 3
No error is found
● Original data bits: 0011
```

Tramas con 1 Error

1.

```
> python .\emisor.py
Introduzca una trama en binario:
1001
● Mensaje codificado: 0011001
> node .\receptor.js
DATA: 0011101 Paridad: 3
Error is found at position: 5
● Corrected data: 0011001
  Extracted data bits: 1001
```

2.

```
> python .\emisor.py
Introduzca una trama en binario:
1010
Mensaje codificado: 1011010
> node .\receptor.js
DATA: 1111010 Paridad: 3
Error is found at position: 2
• Corrected data: 1011010
Extracted data bits: 1010
```

3.

```
> python .\emisor.py
Introduzca una trama en binario:
00011100
Mensaje codificado: 000100101100
> node .\receptor.js
DATA: 000100111100 Paridad: 5
Error is found at position: 2
• Corrected data: 010100111100
Extracted data bits: 00011100
```

Tramas con 2 Errores:

1.

```
> python .\emisor.py
Introduzca una trama en binario:
1001
Mensaje codificado: 0011001
> node .\receptor.js
DATA: 0010101 Paridad: 3
Error is found at position: 4
• Corrected data: 0011101
Extracted data bits: 1101
```

2.

```
> python .\emisor.py
Introduzca una trama en binario:
0011
Mensaje codificado: 1000011
> node .\receptor.js
DATA: 1001111 Paridad: 3
Error is found at position: 4
• Corrected data: 1000111
Extracted data bits: 0111
```

```
> python .\emisor.py
Introduzca una trama en binario:
1010
Mensaje codificado: 1011010
> node .\receptor.js
DATA: 1011100 Paridad: 3
Error is found at position: 6
Corrected data: 1011110
Extracted data bits: 1110
```

3.

Tramas Modificadas pero funcionales:

```
> python .\emisor.py
Introduzca una trama en binario:
0000
Mensaje codificado: 00000000
> node .\receptor.js
DATA: 1111111 Paridad: 3
No error is found
● Original data bits: 1111
```

1.

Discusión

En cuanto a los resultados, ambos algoritmos implementados fueron capaces de detectar eficientemente errores aleatorios de 1 bit introducidos manualmente en diferentes posiciones de las tramas de prueba.

Con CRC-32 no se presentaron problemas en la detección de errores para ninguna de las tramas de prueba. Esto era de esperarse dado que CRC-32 es altamente efectivo detectando errores bursts y aleatorios para tramas de cualquier longitud.

Por su parte, Hamming también detectó los errores en las tramas de longitud adecuada que cumplieran con la relación $(m+r+1) \leq 2^r$. Adicionalmente cuando solo era 1 el error el algoritmo era capaz de identificarlo y corregirlo. Sin embargo al haber dos errores es capaz de encontrar el primero y corregirlo sin embargo para el segundo no puede corregirlo porque el resultado de los bits de paridad es solo un número por lo que solo podemos detectar un error a la vez.

En cuanto a la detección de patrones de error específicos, se probó una trama donde se invirtieron todos los bits de paridad calculados por Hamming. De esta manera, el algoritmo fue engañado y no detectó la presencia de errores. Esto revela una vulnerabilidad de Hamming ante ciertos patrones de error que debería ser mitigada en implementaciones prácticas.

Por el contrario, el CRC-32 mostró ser robusto incluso ante este patrón de error al utilizar un polinomio generador más complejo. Por lo tanto, CRC-32 es menos susceptible a fallas ante ciertos patrones de bits errados.

Comentario grupal sobre el tema (errores)

Consideramos que el tema de detección y corrección de errores es fundamental en los sistemas de comunicación actuales. Los canales están sujetos a ruido e interferencia que introducen errores aleatorios en los datos transmitidos. Si estos errores no son manejados adecuadamente, se degradan significativamente las tasas de transferencia efectiva y la calidad de los enlaces.

Mediante este laboratorio pudimos comprender las diferencias entre los esquemas de detección y corrección de errores. Vimos que algoritmos como CRC son muy utilizados en la práctica por su eficiencia en la detección de errores aleatorios. Otros como Hamming, aunque más complejos, permiten incluso la corrección de errores.

Queda claro que la elección del algoritmo dependerá de las necesidades de cada aplicación. En sistemas tolerantes a retardos como transferencia de archivos, es preferible utilizar CRC para la detección. En aplicaciones críticas como control industrial, vale la pena implementar Hamming u otros esquemas de corrección.

Creemos que es importante como ingenieros entender estos conceptos para diseñar sistemas de comunicación confiables. El reto actual es desarrollar algoritmos de detección/corrección que sean eficientes incluso en los enlaces inalámbricos cada vez más utilizados.

Conclusiones

1. El algoritmo de CRC-32 es muy efectivo detectando errores en tramas de cualquier longitud, mientras que Hamming está limitado a tramas de cierta longitud que cumplan la relación $(m + r + 1) \leq 2^r$.
2. Hamming puede no solo detectar sino también corregir errores, mientras que CRC-32 solo detecta la presencia de errores. Sin embargo, la capacidad de corrección de Hamming es limitada a 1 bit por trama.
3. La implementación de CRC-32 resulta más sencilla que Hamming, ya que solo requiere realizar una división modular utilizando un polinomio generador estándar. Hamming requiere calcular y manipular la paridad de los bits de datos y redundancia.
4. Las pruebas realizadas demuestran que ambos algoritmos detectan eficientemente errores aleatorios de 1 bit en las tramas de datos. Sin embargo, CRC-32 es más robusto ante patrones de error que pueden pasar desapercibidos por Hamming.
5. Tanto CRC-32 como Hamming tienen una probabilidad de falsa alarma muy baja (indicar un error cuando no lo hay). Esto los hace confiables para determinar si una trama tiene errores antes de descartarla o corregirla.

Redes
Laboratorio 2
Juan Angel Carrera 20593
José Mariano Reyes 20074

Referencias:

- Wikipedia Contributors. (2023, July 24). Cyclic redundancy check. Wikipedia; Wikimedia Foundation. https://en.wikipedia.org/wiki/Cyclic_redundancy_check
- CRC32: Verificación de Redundancia Cíclica. (2019). Jc-Mouse.net. <https://www.jc-mouse.net/java/crc32-verificacion-de-redundancia-ciclica>