

UNIVERSIDAD DEL VALLE DE GUATEMALA

CC3067 - REDES

Sección 11

MIGUEL NOVELLA LINARES



Laboratorio 2 - Primera parte

Esquemas de detección y corrección de errores

Isabella Miralles #22293

Ruth de León #22428

Guatemala, julio 2025

Corrección de errores

En este laboratorio, para la corrección de errores se implementó el algoritmo Hamming (7,4). Este permite codificar una trama de 4 bits en una de 7 bits añadiendo 3 bits de paridad para la detección y corrección de errores.

Por medio de pruebas controladas y utilizando el mismo conjunto de mensajes base, se implementó el emisor en c++ y el receptor en python.

Escenarios de prueba

Cada mensaje se probó en tres escenarios

- Trama sin errores
- Trama con un solo bit alterado
- Trama con dos bits alterados

```
===== PRUEBAS AUTOMATIZADAS DEL RECEPTOR HAMMING (7,4) =====  
  
Mensaje original: 1011  
Trama original : 0111011  
  
[1] Sin errores:  
Se detectó y corrigió un error en la posición 4.  
Trama corregida: 0110011  
Datos extraídos: 1011  
  
[2] Con 1 error:  
Trama modificada: 0101011  
Se detectó y corrigió un error en la posición 7.  
Trama corregida: 0101010  
Datos extraídos: 0010  
  
[3] Con 2 errores:  
Trama modificada: 0101010  
No se detectaron errores. Datos: 0010
```

Mensaje original: 0001
Trama original : 1000001

[1] Sin errores:
Se detectó y corrigió un error en la posición 6.
Trama corregida: 1000011
Datos extraídos: 0011

[2] Con 1 error:
Trama modificada: 1010001
Se detectó y corrigió un error en la posición 5.
Trama corregida: 1010101
Datos extraídos: 1101

[3] Con 2 errores:
Trama modificada: 1010000
Se detectó y corrigió un error en la posición 2.
Trama corregida: 1110000
Datos extraídos: 1000

Mensaje original: 1111
Trama original : 0001111

[1] Sin errores:
No se detectaron errores. Datos: 0111

[2] Con 1 error:
Trama modificada: 0011111
Se detectó y corrigió un error en la posición 3.
Trama corregida: 0001111
Datos extraídos: 0111

[3] Con 2 errores:
Trama modificada: 0011110
Se detectó y corrigió un error en la posición 4.
Trama corregida: 0010110

Datos extraídos: 1110

Mensaje original: 0100

Trama original : 1001100

[1] Sin errores:

No se detectaron errores. Datos: 0100

[2] Con 1 error:

Trama modificada: 1011100

Se detectó y corrigió un error en la posición 3.

Trama corregida: 1001100

Datos extraídos: 0100

[3] Con 2 errores:

Trama modificada: 1011101

Se detectó y corrigió un error en la posición 4.

Trama corregida: 1010101

Datos extraídos: 1101

Mensaje original: 1100

Trama original : 0111100

[1] Sin errores:

No se detectaron errores. Datos: 1100

[2] Con 1 error:

Trama modificada: 0101100

Se detectó y corrigió un error en la posición 3.

Trama corregida: 0111100

Datos extraídos: 1100

[3] Con 2 errores:

Trama modificada: 0101101

Se detectó y corrigió un error en la posición 4.

Trama corregida: 0100101

```
Datos extraídos: 0101

Mensaje original: 0011
Trama original : 1000011

[1] Sin errores:
No se detectaron errores. Datos: 0011

[2] Con 1 error:
No se detectaron errores. Datos: 0011

[2] Con 1 error:

[2] Con 1 error:
[2] Con 1 error:
Trama modificada: 1010011
Trama modificada: 1010011
Se detectó y corrigió un error en la posición 3.
Trama corregida: 1000011
Datos extraídos: 0011

[3] Con 2 errores:
Trama modificada: 1010010
Se detectó y corrigió un error en la posición 4.
Trama corregida: 1011010
Datos extraídos: 1010
```

Preguntas

¿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?

Si, ya que el algoritmo solo corrige un error de bit. Al momento de alterar dos bits en posiciones específicas, es posible que el algoritmo no lo detecte o que detecte el error, pero corrija el bit incorrecto, esto genera datos corruptos sin advertencia.

Ventajas

- Corrige un error de bit con solo 3 bits extra
- Bajo costo computacional
- Implementación sencilla

Desventajas

- No detecta múltiples errores con certeza
- No es escalable (Solo 4 bits de datos por trama)
- Vulnerable a errores simétricos de 2 bits

Detección de errores

En esta sección se implementó el algoritmo CRC-32 (Cyclic Redundancy Check) para detectar errores en tramas binarias. Este método permite verificar la integridad de los datos transmitidos usando un polinomio generador estándar de 32 bits: Polinomio estándar CRC-32: $0x04C11DB7$ ($x^{32} + x^6 + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^8 + x^7 + x^5 + x^4 + x^2 + x + 1$)

Emisor

1. Solicita una trama binaria al usuario ("11010101").
2. Se calcula el CRC-32 de la trama, agregando 32 bits de ceros al final antes de aplicar el algoritmo.
3. El resultado del CRC se concatena al final de la trama original para formar el mensaje transmitido.



```
crc32_emisor.py
crc32_emisor.py > ...
1 def crc32_emisor(data_binaria):
2     polinomio = 0x04C11DB7
3     data = data_binaria + '0' * 32
4     data = list(map(int, data))
5
6     for i in range(len(data_binaria)):
7         if data[i] == 1:
8             for j in range(32):
9                 data[i + j] ^= (polinomio >> (32 - j)) & 1
10
11     crc = ''.join(map(str, data[-32:]))
12     return data_binaria + crc
13
14 # Ejecución de prueba
15 if __name__ == "__main__":
16     mensaje = "11010011101100"
17     resultado = crc32_emisor(mensaje)
18     print("Trama con CRC:", resultado)
19
```

SALIDA PROBLEMAS TERMINAL PUERTOS

```
PS C:\REDES\lab2> py crc32_emisor.py
Trama con CRC: 1101001110110010000101011100110011000110000001
PS C:\REDES\lab2>
```

Receptor

```
1  #include <iostream>
2  #include <string>
3  using namespace std;
4
5  unsigned int crc32_receptor(const string& input) {
6      unsigned int poly = 0x04C11DB7;
7      unsigned int crc = 0;
8      for (char bit : input) {
9          crc ^= ((bit - '0') << 31);
10         for (int i = 0; i < 8; ++i) {
11             if (crc & 0x80000000)
12                 crc = (crc << 1) ^ poly;
13             else
14                 crc <<= 1;
15         }
16     }
17     return crc;
18 }
19
20 int main() {
21     string trama;
22     cout << "Ingrese la trama con CRC: ";
23     cin >> trama;
24     if (crc32_receptor(trama) == 0)
25         cout << "Trama valida: sin errores." << endl;
26     else
27         cout << "Error detectado: trama invalida." << endl;
28     return 0;
29 }
```

TERMINAL

```
PS C:\REDES\lab2> g++ receptor.cpp -o receptor
PS C:\REDES\lab2> ./receptor
Ingrese la trama con CRC: 1101001110110010000101011100110011000110000001
Trama valida: sin errores.
```

Escenario de prueba:

Mensaje base utilizado: 11010011101100

| Tipo de prueba | Trama entregada | Resultado |
|------------------|--|-----------------|
| 2 bits cambiados | 110100111011001000010101110011001100110000000 | Error detectado |
| 1 bit cambiado | 1101001110110010000101011100110011000110000000 | Error detectado |
| Sin cambios | 1101001110110010000101011100110011000110000001 | Trama válida |

Pregunta

¿Es posible manipular los bits de tal forma que el algoritmo seleccionado no sea capaz de detectar el error?

Es muy poco probable que un error no sea detectado con CRC-32. Está diseñado para detectar: Todos los errores de 1 bit, todos los errores de 2 bits y todas las ráfagas de hasta 32 bits.

Básicamente pueden existir colisiones (mensajes diferentes con el mismo CRC), aunque su probabilidad es extremadamente baja.

Ventaja

- Alta capacidad de detección de errores múltiples.
- Uso extendido en redes y almacenamiento.
- Buena eficiencia en hardware/software optimizado

Desventaja

- Alta capacidad de detección de errores múltiples.
- Uso extendido en redes y almacenamiento.
- Buena eficiencia en hardware/software optimizado

Enlace del repositorio

<https://github.com/Isabella-22293/Lab2-Redes.git>