Universidad del Valle de Guatemala
Facultad de Ingeniería
Departamento de Ciencias de la Computación
CC3067 - Redes

Laboratorio #2
Parte 1 y 2

17/08/2025
Sofía Velásquez, 22049
José Rodrigo Marchena, 22398
Diego García, 22404
Semestre II - 2025

Repositorio: https://github.com/Sofiamishel2003/Esquemas-de-deteccio-n-y-correccio-n.git

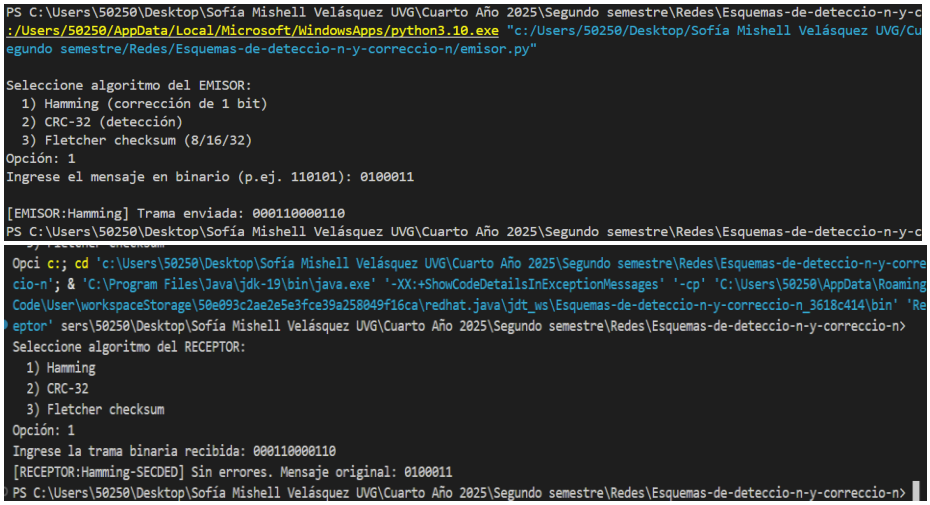
Parte 1

Escenarios de pruebas. Para los tres algoritmos evaluar:

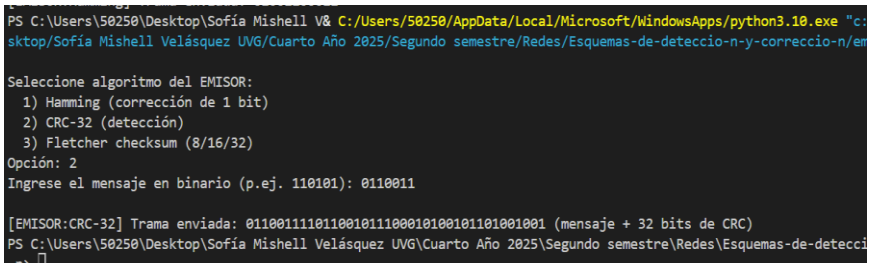
- **Sin Errores**

  Enviar un mensaje al emisor, copiar el mensaje generado por este y proporcionarlo tal cual, al receptor, el cual debe mostrar los mensajes originales (ya que ningún bit sufrió un cambio). Realizar esto para tres mensajes distintos con distinta longitud.

1. Mensaje: 0100011
   - Hamming



   - CRC – 32



   - Fletcher

```
[EMISOR:CRC-32] Trama enviada: 01100111101100101110001010010110100l001 (mensaje + 32 bits de CRC)
PS C:\Users\50250\Desktop\Sofía Mishell Velásquez UVG\Cuarto Año 2025\Segundo semestre\Redes\Esquemas-de-deteccio-
-n> & C:/Users/50250/AppData/Local/Microsoft/WindowsApps/python3.10.exe "c:/Users/50250/Desktop/Sofía Mishell Velá
rto Año 2025/Segundo semestre/Redes/Esquemas-de-deteccio-n-y-correccio-n/emisor.py"

Seleccione algoritmo del EMISOR:
  1) Hamming (corrección de 1 bit)
  2) CRC-32 (detección)
  3) Fletcher checksum (8/16/32)
Opción: 3
Ingrese el mensaje en binario (p.ej. 110101): 0110011
Seleccione Fletcher (8,16,32): 16

[Nota] Para el cálculo se aplicó padding de 1 bit(s) 0 a la IZQUIERDA (agrupación en 8 bits).

[EMISOR:Fletcher-16] Checksum = 0011010100110100
[EMISOR:Fletcher-16] Trama enviada: 01100110011010100110100 (mensaje + 16 bits de checksum)
PS C:\Users\50250\Desktop\Sofía Mishell Velásquez UVG\Cuarto Año 2025\Segundo semestre\Redes\Esquemas-de-deteccio-
-n>
```

```
PS C:\Users\50250\Desktop\Sofía Mishell Velásquez UVG\Cuarto Año 2025\Segundo semestre\Redes\Esquema
-n>  c:; cd 'c:\Users\50250\Desktop\Sofía Mishell Velásquez UVG\Cuarto Año 2025\Segundo semestre\Red
-correccio-n'; & 'C:\Program Files\Java\jdk-19\bin\java.exe' '-XX:+ShowCodeDetailsInExceptionMessage
pData\Roaming\Code\User\workspaceStorage\50e093c2ae2e5e3fce39a258049f16ca\redhat.java\jdt_ws\Esquema
-n_3618c414\bin' 'Receptor'
Seleccione algoritmo del RECEPTOR:
  1) Hamming
  2) CRC-32
  3) Fletcher checksum
Opción: 3
Ingrese la trama binaria recibida: 0110011001101010011010
Seleccione Fletcher (8,16,32): 16
[RECEPTOR:Fletcher-16] Sin errores. Mensaje original: 0110011
PS C:\Users\50250\Desktop\Sofía Mishell Velásquez UVG\Cuarto Año 2025\Segundo semestre\Redes\Esquema
```

2. Mensaje: 10101010111101011010111010111000110100100110101011O1

   - Hamming

```
Seleccione algoritmo del EMISOR:
  1) Hamming (corrección de 1 bit)
  2) CRC-32 (detección)
  3) Fletcher checksum (8/16/32)
Opción: 1
Ingrese el mensaje en binario (p.ej. 110101): 10101010111101011010111010111000110100100110101011O1

[EMISOR:Hamming] Trama enviada: 111001011010111010101101011101001110001101001001101010111011
```

```
Seleccione algoritmo del RECEPTOR:
  1) Hamming
  2) CRC-32
  3) Fletcher checksum
Opción: 1
Ingrese la trama binaria recibida: 111001011010111010101101011101001110001101001001101010111011
[RECEPTOR:Hamming-SECDED] Sin errores. Mensaje original: 10101010111101011010111010111000110100100110101011O1
```

   - CRC-32

```
Seleccione algoritmo del EMISOR:
  1) Hamming (corrección de 1 bit)
  2) CRC-32 (detección)
  3) Fletcher checksum (8/16/32)
Opción: 2
Ingrese el mensaje en binario (p.ej. 110101): 10101010111101011010111010111000110100100110101011O1

[EMISOR:CRC-32] Trama enviada: 10101010111101011010111010111000110100100110101011010000101101010010001101011000101l (mensaje + 32 bits de CRC)
```

```
Seleccione algoritmo del RECEPTOR:
  1) Hamming
  2) CRC-32
  3) Fletcher checksum
Opción: 2
Ingrese la trama binaria recibida: 10101010111101011010111010111000110100100110101011010000101101010010001101011000101l
[RECEPTOR:CRC-32] Sin errores. Mensaje original: 10101010111101011010111010111000110100100110101011O1
```

- Fletcher

```
Seleccione algoritmo del EMISOR:
  1) Hamming (corrección de 1 bit)
  2) CRC-32 (detección)
  3) Fletcher checksum (8/16/32)
Opción: 3
Ingrese el mensaje en binario (p.ej. 110101): 10101010111101011010111010111000110100100110101011010
Seleccione Fletcher (8,16,32): 16

[Nota] Para el cálculo se aplicó padding de 4 bit(s) 0 a la IZQUIERDA (agrupación en 8 bits).

[EMISOR:Fletcher-16] Checksum = 1000001001100010
[EMISOR:Fletcher-16] Trama enviada: 1010101011110101101011101011100011010010011010101101100000100110001010 (mensaje + 16 bits de checksum)
```

```
Seleccione algoritmo del RECEPTOR:
  1) Hamming
  2) CRC-32
  3) Fletcher checksum
Opción: 3
Ingrese la trama binaria recibida: 1010101011110101101011101011100011010010011010101101100000100110001010
Seleccione Fletcher (8,16,32): 16
[RECEPTOR:Fletcher-16] Sin errores. Mensaje original: 1010101011110101101011101011100011010010011010101101
```

3. Mensaje: 11011001101100110100
   - Hamming

```
PS C:\Users\JM\Documents\Redes\Esquemas-de-deteccio-n-y-correccio-n> python emisor.py

Seleccione algoritmo del EMISOR:
  1) Hamming (corrección de 1 bit)
  2) CRC-32 (detección)
  3) Fletcher checksum (8/16/32)
Opción: 1
Ingrese el mensaje en binario (p.ej. 110101): 11011001101100110100

[EMISOR:Hamming] Trama enviada: 101010101001101010011101000
PS C:\Users\JM\Documents\Redes\Esquemas-de-deteccio-n-y-correccio-n>
```

```
PS C:\Users\JM\Documents\Redes\Esquemas-de-deteccio-n-y-correccio-n> java .\Receptor.java
Seleccione algoritmo del RECEPTOR:
  1) Hamming
  2) CRC-32
  3) Fletcher checksum
Opción: 1
Ingrese la trama binaria recibida: 101010101001101010011101000
[RECEPTOR:Hamming-SECDED] Sin errores. Mensaje original: 11011001101100110100
PS C:\Users\JM\Documents\Redes\Esquemas-de-deteccio-n-y-correccio-n>
```

   - CRC-32

```
PS C:\Users\JM\Documents\Redes\Esquemas-de-deteccio-n-y-correccio-n> python emisor.py

Seleccione algoritmo del EMISOR:
  1) Hamming (corrección de 1 bit)
  2) CRC-32 (detección)
  3) Fletcher checksum (8/16/32)
Opción: 2
Ingrese el mensaje en binario (p.ej. 110101): 11011001101100110100

[EMISOR:CRC-32] Trama enviada: 11011001101100110100101001010011011100111111011100101 (mensaje + 32 bits de CRC)
PS C:\Users\JM\Documents\Redes\Esquemas-de-deteccio-n-y-correccio-n>
```

Sofía Velásquez, 22049
José Rodrigo Marchena, 22398
Diego García, 22404


```
PS C:\Users\JM\Documents\Redes\Esquemas-de-deteccio-n-y-correccio-n> java .\Receptor.java
Seleccione algoritmo del RECEPTOR:
  1) Hamming
  2) CRC-32
  3) Fletcher checksum
Opción: 2
Ingrese la trama binaria recibida: 1101100110110011010010100101001101110011111011100101
[RECEPTOR:CRC-32] Sin errores. Mensaje original: 11011001101100110100
PS C:\Users\JM\Documents\Redes\Esquemas-de-deteccio-n-y-correccio-n>
```

- Fletcher (16)

```
PS C:\Users\JM\Documents\Redes\Esquemas-de-deteccio-n-y-correccio-n> python emisor.py

Seleccione algoritmo del EMISOR:
  1) Hamming (corrección de 1 bit)
  2) CRC-32 (detección)
  3) Fletcher checksum (8/16/32)
Opción: 3
Ingrese el mensaje en binario (p.ej. 110101): 11011001101100110100
Seleccione Fletcher (8,16,32): 16

[Nota] Para el cálculo se aplicó padding de 4 bit(s) 0 a la IZQUIERDA (agrupación en 8 bits).

[EMISOR:Fletcher-16] Checksum = 1001011011011101
[EMISOR:Fletcher-16] Trama enviada: 1101100110110011010010010110110111011101 (mensaje + 16 bits de checksum)
PS C:\Users\JM\Documents\Redes\Esquemas-de-deteccio-n-y-correccio-n>
```

```
PS C:\Users\JM\Documents\Redes\Esquemas-de-deteccio-n-y-correccio-n> java .\Receptor.java
Seleccione algoritmo del RECEPTOR:
  1) Hamming
  2) CRC-32
  3) Fletcher checksum
Opción: 3
Ingrese la trama binaria recibida: 1101100110110011010010010110110111011101
Seleccione Fletcher (8,16,32): 16
[RECEPTOR:Fletcher-16] Sin errores. Mensaje original: 11011001101100110100
PS C:\Users\JM\Documents\Redes\Esquemas-de-deteccio-n-y-correccio-n>
```

- **Un Error**:
  Enviar un mensaje al emisor, copiar el mensaje generado por este y cambiar un bit cualquiera antes de proporcionarlo al receptor. Si el algoritmo es de detección debe mostrar que se detectó un error y que se descarta el mensaje. Si el algoritmo es de corrección debe corregir el bit, indicar su posición y mostrar el mensaje original. Realizar esto para tres mensajes distintos con distinta longitud.

1. Mensaje: 0100011
   - Hamming

```
PS C:\Users\50250\Desktop\Sofía Mishell Velásquez UVG\Cuarto Año 2025\Segundo semestre\Redes\Esquemas-de-deteccio-n-y-c
:/Users/50250/AppData/Local/Microsoft/WindowsApps/python3.10.exe "c:/Users/50250/Desktop/Sofía Mishell Velásquez UVG/Cu
egundo semestre/Redes/Esquemas-de-deteccio-n-y-correccio-n/emisor.py"

Seleccione algoritmo del EMISOR:
  1) Hamming (corrección de 1 bit)
  2) CRC-32 (detección)
  3) Fletcher checksum (8/16/32)
Opción: 1
Ingrese el mensaje en binario (p.ej. 110101): 0100011

[EMISOR:Hamming] Trama enviada: 000110000110
PS C:\Users\50250\Desktop\Sofía Mishell Velásquez UVG\Cuarto Año 2025\Segundo semestre\Redes\Esquemas-de-deteccio-n-y-c
```

```
; cd 'c:\Users\50250\Desktop\Sofía Mishell Velásquez UVG\Cuarto Año 2025\Segundo semestre\Redes\Esquema
 & 'C:\Program Files\Java\jdk-19\bin\java.exe' '-XX:+ShowCodeDetailsInExceptionMessages' '-cp' 'C:\User
er\workspaceStorage\50e093c2ae2e5e3fce39a258049f16ca\redhat.java\jdt_ws\Esquemas-de-deteccio-n-y-correc

Seleccione algoritmo del RECEPTOR:
  1) Hamming
  2) CRC-32
  3) Fletcher checksum
Opción: 1
Ingrese la trama binaria recibida: 000010000110
[RECEPTOR:Hamming-SECDED] Se corrigió 1 bit en posición 4. Mensaje corregido: 0100011
PS C:\Users\50250\Desktop\Sofía Mishell Velásquez UVG\Cuarto Año 2025\Segundo semestre\Redes\Esquemas-d
```

- CRC-32

```
PS C:\Users\50250\Desktop\Sofía Mishell Velásquez UVG\Cuarto Año 2025\Segundo semestre\Redes\Esquemas-
-n> & C:/Users/50250/AppData/Local/Microsoft/WindowsApps/python3.10.exe "c:/Users/50250/Desktop/Sofía
rto Año 2025/Segundo semestre/Redes/Esquemas-de-deteccio-n-y-correccio-n/emisor.py"

Seleccione algoritmo del EMISOR:
  1) Hamming (corrección de 1 bit)
  2) CRC-32 (detección)
  3) Fletcher checksum (8/16/32)
Opción: 2
Ingrese el mensaje en binario (p.ej. 110101): 0100011

[EMISOR:CRC-32] Trama enviada: 01000111001010101100000100100000111001 (mensaje + 32 bits de CRC)
PS C:\Users\50250\Desktop\Sofía Mishell Velásquez UVG\Cuarto Año 2025\Segundo semestre\Redes\Esquemas-
```

```
PS C:\Users\50250\Desktop\Sofía Mishell Velásquez UVG\Cuarto Año 2025\Segundo semestre\Redes\Esquemas-
-n>  c:; cd 'c:\Users\50250\Desktop\Sofía Mishell Velásquez UVG\Cuarto Año 2025\Segundo semestre\Redes
-correccio-n'; & 'C:\Program Files\Java\jdk-19\bin\java.exe' '-XX:+ShowCodeDetailsInExceptionMessages'
pData\Roaming\Code\User\workspaceStorage\50e093c2ae2e5e3fce39a258049f16ca\redhat.java\jdt_ws\Esquemas-
-n_3618c414\bin' 'Receptor'
Seleccione algoritmo del RECEPTOR:
  1) Hamming
  2) CRC-32
  3) Fletcher checksum
Opción: 2
Ingrese la trama binaria recibida: 01000111001010101100000100100000111001
[RECEPTOR:CRC-32] Se detectaron errores. Mensaje descartado.
PS C:\Users\50250\Desktop\Sofía Mishell Velásquez UVG\Cuarto Año 2025\Segundo semestre\Redes\Esquemas-
```

- Fletcher

```
PS C:\Users\50250\Desktop\Sofía Mishell Velásquez UVG\Cuarto Año 2025\Segundo semestre\Redes\Esquemas
-n> & C:/Users/50250/AppData/Local/Microsoft/WindowsApps/python3.10.exe "c:/Users/50250/Desktop/Sofía
rto Año 2025/Segundo semestre/Redes/Esquemas-de-deteccio-n-y-correccio-n/emisor.py"

Seleccione algoritmo del EMISOR:
  1) Hamming (corrección de 1 bit)
  2) CRC-32 (detección)
  3) Fletcher checksum (8/16/32)
Opción: 3
Ingrese el mensaje en binario (p.ej. 110101): 0100011
Seleccione Fletcher (8,16,32): 32

[Nota] Para el cálculo se aplicó padding de 9 bit(s) 0 a la IZQUIERDA (agrupación en 16 bits).

[EMISOR:Fletcher-32] Checksum = 00000000001001010000000000100100
[EMISOR:Fletcher-32] Trama enviada: 0100011000000000010010100000000000100100 (mensaje + 32 bits de che
PS C:\Users\50250\Desktop\Sofía Mishell Velásquez UVG\Cuarto Año 2025\Segundo semestre\Redes\Esquemas
```

```
PS C:\Users\50250\Desktop\Sofía Mishell Velásquez UVG\Cuarto Año 2025\Segundo semestre\Redes\Esquemas
-n>  c:; cd 'c:\Users\50250\Desktop\Sofía Mishell Velásquez UVG\Cuarto Año 2025\Segundo semestre\Rede
-correccio-n'; & 'C:\Program Files\Java\jdk-19\bin\java.exe' '-XX:+ShowCodeDetailsInExceptionMessages
pData\Roaming\Code\User\workspaceStorage\50e093c2ae2e5e3fce39a258049f16ca\redhat.java\jdt_ws\Esquemas
-n_3618c414\bin' 'Receptor'
Seleccione algoritmo del RECEPTOR:
  1) Hamming
  2) CRC-32
  3) Fletcher checksum
Opción: 3
Ingrese la trama binaria recibida: 0100011000000000010000100000000000100100
Seleccione Fletcher (8,16,32): 32
[RECEPTOR:Fletcher-32] Se detectaron errores. Mensaje descartado.
PS C:\Users\50250\Desktop\Sofía Mishell Velásquez UVG\Cuarto Año 2025\Segundo semestre\Redes\Esquemas
```

2. Mensaje: 10101010111101011010111010111000110100100110101011 01

   - Hamming

```
Seleccione algoritmo del EMISOR:
  1) Hamming (corrección de 1 bit)
  2) CRC-32 (detección)
  3) Fletcher checksum (8/16/32)
Opción: 1
Ingrese el mensaje en binario (p.ej. 110101): 10101010111101011010111010111000110100100110101011 01

[EMISOR:Hamming] Trama enviada: 11100101101011101010110101110100111000110100100110101011011
```

```
Seleccione algoritmo del RECEPTOR:
  1) Hamming
  2) CRC-32
  3) Fletcher checksum
Opción: 1
Ingrese la trama binaria recibida: 11100101101011101010110001110100111000110100100110101011011
[RECEPTOR:Hamming-SECDED] Se corrigió 1 bit en posición 24. Mensaje corregido: 10101010111101011010111010111000110100100110101011 01
```

   - CRC-32

```
Seleccione algoritmo del EMISOR:
  1) Hamming (corrección de 1 bit)
  2) CRC-32 (detección)
  3) Fletcher checksum (8/16/32)
Opción: 2
Ingrese el mensaje en binario (p.ej. 110101): 10101010111101011010111010111000110100100110101011 01

[EMISOR:CRC-32] Trama enviada: 10101010111101011010111010111000110100100110101011010000101101010010001101011000101 1 (mensaje + 32 bits de CRC)
```

```
Seleccione algoritmo del RECEPTOR:
  1) Hamming
  2) CRC-32
  3) Fletcher checksum
Opción: 2
Ingrese la trama binaria recibida: 10101010111101011000111010111000110100100110101011010000101101010010001101011000101 1
[RECEPTOR:CRC-32] Se detectaron errores. Mensaje descartado.
```

- Fletcher

```
Seleccione algoritmo del EMISOR:
  1) Hamming (corrección de 1 bit)
  2) CRC-32 (detección)
  3) Fletcher checksum (8/16/32)
Opción: 3
Ingrese el mensaje en binario (p.ej. 110101): 1010101011110101101011101011000110100100110101011011000001001100010
Seleccione Fletcher (8,16,32): 16

[Nota] Para el cálculo se aplicó padding de 4 bit(s) 0 a la IZQUIERDA (agrupación en 8 bits).

[EMISOR:Fletcher-16] Checksum = 1000001001100010
[EMISOR:Fletcher-16] Trama enviada: 1010101011110101101011101011000110100100110101011011000001001100010 (mensaje + 16 bits de checksum)
```

```
Seleccione algoritmo del RECEPTOR:
  1) Hamming
  2) CRC-32
  3) Fletcher checksum
Opción: 3
Ingrese la trama binaria recibida: 1010101011110101100011101011100011010010011010101101100000100110001O
Seleccione Fletcher (8,16,32): 16
[RECEPTOR:Fletcher-16] Se detectaron errores. Mensaje descartado.
```

3. Mensaje: 00001000010000
   - Hamming

```
PS C:\Users\JM\Documents\Redes\Esquemas-de-deteccio-n-y-correccio-n> python emisor.py

Seleccione algoritmo del EMISOR:
  1) Hamming (corrección de 1 bit)
  2) CRC-32 (detección)
  3) Fletcher checksum (8/16/32)
Opción: 1
Ingrese el mensaje en binario (p.ej. 110101): 00001000010000

[EMISOR:Hamming] Trama enviada: 11010000100001000001
PS C:\Users\JM\Documents\Redes\Esquemas-de-deteccio-n-y-correccio-n>
```

```
PS C:\Users\JM\Documents\Redes\Esquemas-de-deteccio-n-y-correccio-n> java .\Receptor.java
Seleccione algoritmo del RECEPTOR:
  1) Hamming
  2) CRC-32
  3) Fletcher checksum
Opción: 1
Ingrese la trama binaria recibida: 11010000100001000000
[RECEPTOR:Hamming-SECDED] Se corrigió el bit de paridad global (p0). Mensaje: 00001000010000
PS C:\Users\JM\Documents\Redes\Esquemas-de-deteccio-n-y-correccio-n>
```

   - CRC-32

```
Seleccione algoritmo del EMISOR:
  1) Hamming (corrección de 1 bit)
  2) CRC-32 (detección)
  3) Fletcher checksum (8/16/32)
Opción: 2
Ingrese el mensaje en binario (p.ej. 110101): 00001000010000

[EMISOR:CRC-32] Trama enviada: 00001000010000111011001110001101000101011111111 (mensaje + 32 bits de CRC)
PS C:\Users\JM\Documents\Redes\Esquemas-de-deteccio-n-y-correccio-n>
```

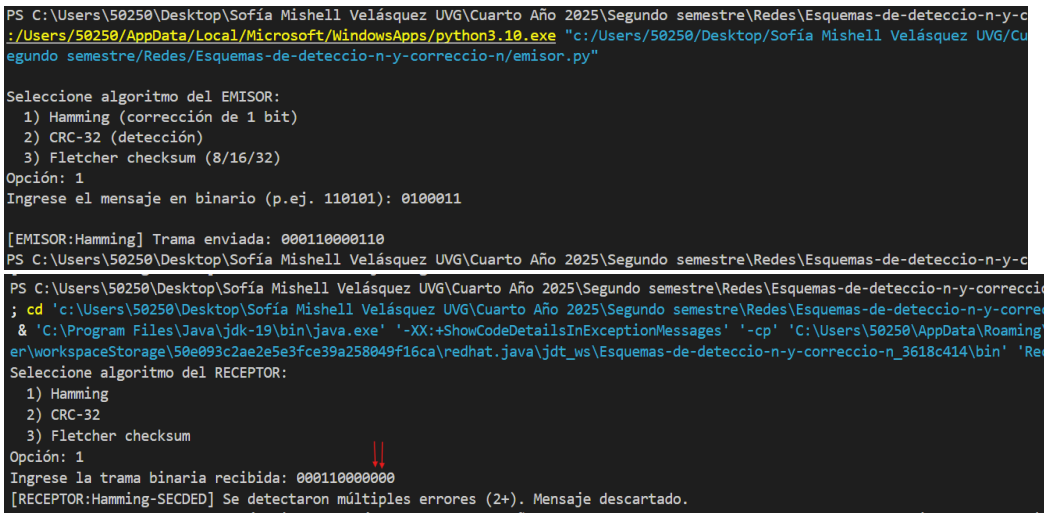
```
PS C:\Users\JM\Documents\Redes\Esquemas-de-deteccio-n-y-correccio-n> java .\Receptor.java
Seleccione algoritmo del RECEPTOR:
  1) Hamming
  2) CRC-32
  3) Fletcher checksum
Opción: 2
Ingrese la trama binaria recibida: 00001000010000111011001110001101000101011111110
[RECEPTOR:CRC-32] Se detectaron errores. Mensaje descartado.
PS C:\Users\JM\Documents\Redes\Esquemas-de-deteccio-n-y-correccio-n>
```

- Fletcher

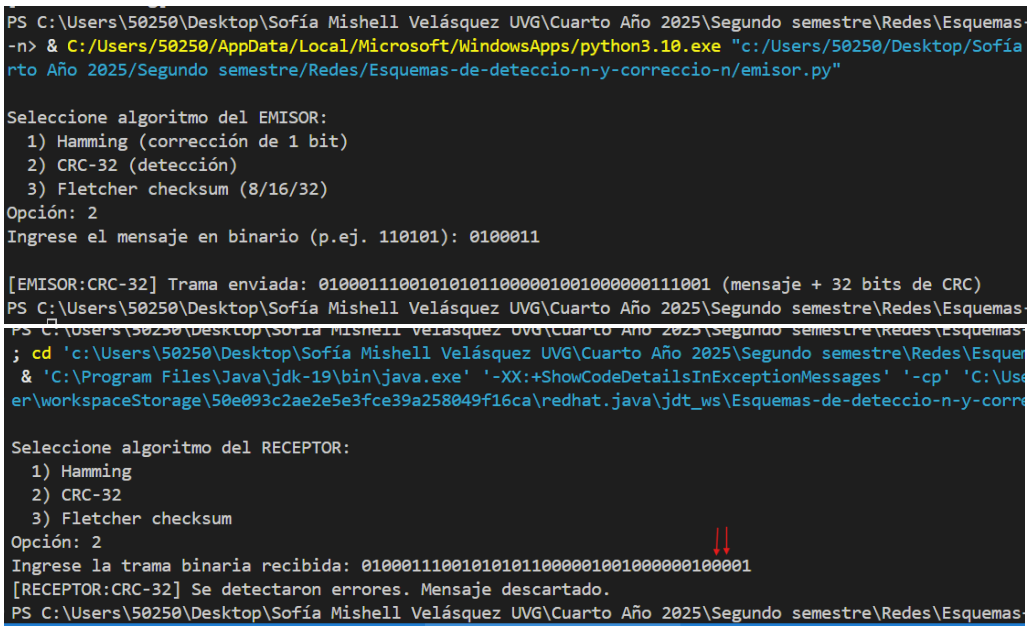
```
PS C:\Users\JM\Documents\Redes\Esquemas-de-deteccio-n-y-correccio-n> python emisor.py

Seleccione algoritmo del EMISOR:
  1) Hamming (corrección de 1 bit)
  2) CRC-32 (detección)
  3) Fletcher checksum (8/16/32)
Opción: 3
Ingrese el mensaje en binario (p.ej. 110101): 00001000010000
Seleccione Fletcher (8,16,32): 16

[Nota] Para el cálculo se aplicó padding de 2 bit(s) 0 a la IZQUIERDA (agrupación en 8 bits).

[EMISOR:Fletcher-16] Checksum = 0001011100010011
[EMISOR:Fletcher-16] Trama enviada: 0000100001000000001011100010011 (mensaje + 16 bits de checksum)
PS C:\Users\JM\Documents\Redes\Esquemas-de-deteccio-n-y-correccio-n>
```

```
PS C:\Users\JM\Documents\Redes\Esquemas-de-deteccio-n-y-correccio-n> java .\Receptor.java
Seleccione algoritmo del RECEPTOR:
  1) Hamming
  2) CRC-32
  3) Fletcher checksum
Opción: 3
Ingrese la trama binaria recibida: 0000100001000000001011100010010
Seleccione Fletcher (8,16,32): 16
[RECEPTOR:Fletcher-16] Se detectaron errores. Mensaje descartado.
PS C:\Users\JM\Documents\Redes\Esquemas-de-deteccio-n-y-correccio-n>
```

- (dos+ errores): Enviar un mensaje al emisor, copiar el mensaje generado por este y cambiar dos o más bits cualesquiera antes de proporcionarlo al receptor. Si el algoritmo es de detección debe mostrar que se detectó un error y que se descarta el mensaje. Si el algoritmo es de corrección y puede corregir más de un error, debe corregir los bits, indicar su posición y mostrar el mensaje original. Realizar esto para tres mensajes distintos con distinta longitud.
  1. Mensaje: 0100011
     - Hamming



- CRC-32



- Fletcher

Sofía Velásquez, 22049
José Rodrigo Marchena, 22398
Diego García, 22404


```
:/Users/50250/AppData/Local/Microsoft/WindowsApps/python3.10.exe "c:/Users/50250/Desktop/Sofía Mishell
egundo semestre/Redes/Esquemas-de-deteccio-n-y-correccio-n/emisor.py"

Seleccione algoritmo del EMISOR:
  1) Hamming (corrección de 1 bit)
  2) CRC-32 (detección)
  3) Fletcher checksum (8/16/32)
Opción: 3
Ingrese el mensaje en binario (p.ej. 110101): 0100011
Seleccione Fletcher (8,16,32): 8

[Nota] Para el cálculo se aplicó padding de 1 bit(s) 0 a la IZQUIERDA (agrupación en 4 bits).

[EMISOR:Fletcher-8] Checksum = 10100110
[EMISOR:Fletcher-8] Trama enviada: 010001110100110 (mensaje + 8 bits de checksum)
PS C:\Users\50250\Desktop\Sofía Mishell Velásquez UVG\Cuarto Año 2025\Segundo semestre\Redes\Esquemas-
```

```
PS C:\Users\50250\Desktop\Sofía Mishell Velásquez UVG\Cuarto Año 2025\Segundo semestre\Redes\Esquemas-
; cd 'c:\Users\50250\Desktop\Sofía Mishell Velásquez UVG\Cuarto Año 2025\Segundo semestre\Redes\Esquem
 & 'C:\Program Files\Java\jdk-19\bin\java.exe' '-XX:+ShowCodeDetailsInExceptionMessages' '-cp' 'C:\Use
er\workspaceStorage\50e093c2ae2e5e3fce39a258049f16ca\redhat.java\jdt_ws\Esquemas-de-deteccio-n-y-corre

Seleccione algoritmo del RECEPTOR:
  1) Hamming
  2) CRC-32
  3) Fletcher checksum
Opción: 3
Ingrese la trama binaria recibida: 010001110100000
Seleccione Fletcher (8,16,32): 8
[RECEPTOR:Fletcher-8] Se detectaron errores. Mensaje descartado.
PS C:\Users\50250\Desktop\Sofía Mishell Velásquez UVG\Cuarto Año 2025\Segundo semestre\Redes\Esquemas-
```

2. Mensaje: 10101010111101011010111010111000110100100110101011O1

   - Hamming

```
Seleccione algoritmo del EMISOR:
  1) Hamming (corrección de 1 bit)
  2) CRC-32 (detección)
  3) Fletcher checksum (8/16/32)
Opción: 1
Ingrese el mensaje en binario (p.ej. 110101): 10101010111101011010111010111000110100100110101011O1

[EMISOR:Hamming] Trama enviada: 111001011010111010101101011101001110001101001001101010111011
```

```
Seleccione algoritmo del RECEPTOR:
  1) Hamming
  2) CRC-32
  3) Fletcher checksum
Opción: 1
Ingrese la trama binaria recibida: 111000011010111010101111011101001110001101001001101010111011
[RECEPTOR:Hamming-SECDED] Se detectaron múltiples errores (2+). Mensaje descartado.
```

   - CRC-32

```
Seleccione algoritmo del EMISOR:
  1) Hamming (corrección de 1 bit)
  2) CRC-32 (detección)
  3) Fletcher checksum (8/16/32)
Opción: 2
Ingrese el mensaje en binario (p.ej. 110101): 10101010111101011010111010111000110100100110101011O1

[EMISOR:CRC-32] Trama enviada: 1010101011110101101011101011100011010010011010101101000010110101001000110101100010111 (mensaje + 32 bits de CRC)
```

```
Seleccione algoritmo del RECEPTOR:
  1) Hamming
  2) CRC-32
  3) Fletcher checksum
Opción: 2
Ingrese la trama binaria recibida: 1010101010110101011111010111000110100100110101011010000101101010010001101011000101111
[RECEPTOR:CRC-32] Se detectaron errores. Mensaje descartado.
```

- Fletcher

```
Seleccione algoritmo del EMISOR:
  1) Hamming (corrección de 1 bit)
  2) CRC-32 (detección)
  3) Fletcher checksum (8/16/32)
Opción: 3
Ingrese el mensaje en binario (p.ej. 110101): 10101010111101011010111010111000110100100110101011
Seleccione Fletcher (8,16,32): 16

[Nota] Para el cálculo se aplicó padding de 4 bit(s) 0 a la IZQUIERDA (agrupación en 8 bits).

[EMISOR:Fletcher-16] Checksum = 1000001001100010
[EMISOR:Fletcher-16] Trama enviada: 1010101011110101101011101011100011010010011010101101100000100110001 0 (mensaje + 16 bits de checksum)
```

```
Seleccione algoritmo del RECEPTOR:
  1) Hamming
  2) CRC-32
  3) Fletcher checksum
Opción: 3
Ingrese la trama binaria recibida: 1010101011110111101011101011100011000010011010101101100000100110001 0
Seleccione Fletcher (8,16,32): 16
[RECEPTOR:Fletcher-16] Se detectaron errores. Mensaje descartado.
```

3. Mensaje: 1111111111101
   - Hamming

```
PS C:\Users\JM\Documents\Redes\Esquemas-de-deteccio-n-y-correccio-n> python emisor.py

Seleccione algoritmo del EMISOR:
  1) Hamming (corrección de 1 bit)
  2) CRC-32 (detección)
  3) Fletcher checksum (8/16/32)
Opción: 1
Ingrese el mensaje en binario (p.ej. 110101): 1111111111101

[EMISOR:Hamming] Trama enviada: 1011111111111111010
PS C:\Users\JM\Documents\Redes\Esquemas-de-deteccio-n-y-correccio-n>
```

```
PS C:\Users\JM\Documents\Redes\Esquemas-de-deteccio-n-y-correccio-n> java .\Receptor.java
Seleccione algoritmo del RECEPTOR:
  1) Hamming
  2) CRC-32
  3) Fletcher checksum
Opción: 1
Ingrese la trama binaria recibida: 0011111111111111011
[RECEPTOR:Hamming-SECDED] Se detectaron múltiples errores (2+). Mensaje descartado.
PS C:\Users\JM\Documents\Redes\Esquemas-de-deteccio-n-y-correccio-n>
```

   - CRC-32

```
PS C:\Users\JM\Documents\Redes\Esquemas-de-deteccio-n-y-correccio-n> python emisor.py

Seleccione algoritmo del EMISOR:
  1) Hamming (corrección de 1 bit)
  2) CRC-32 (detección)
  3) Fletcher checksum (8/16/32)
Opción: 2
Ingrese el mensaje en binario (p.ej. 110101): 1111111111101

[EMISOR:CRC-32] Trama enviada: 11111111111010001010110100011110111011110000 1 (mensaje + 32 bits de CRC)
PS C:\Users\JM\Documents\Redes\Esquemas-de-deteccio-n-y-correccio-n>
```

```
PS C:\Users\JM\Documents\Redes\Esquemas-de-deteccio-n-y-correccio-n> java .\Receptor.java
Seleccione algoritmo del RECEPTOR:
  1) Hamming
  2) CRC-32
  3) Fletcher checksum
Opción: 2
Ingrese la trama binaria recibida: 0111111111101000101011010001111011101111000001
[RECEPTOR:CRC-32] Se detectaron errores. Mensaje descartado.
PS C:\Users\JM\Documents\Redes\Esquemas-de-deteccio-n-y-correccio-n>
```

   - Fletcher

```
PS C:\Users\JM\Documents\Redes\Esquemas-de-deteccio-n-y-correccio-n> python emisor.py

Seleccione algoritmo del EMISOR:
  1) Hamming (corrección de 1 bit)
  2) CRC-32 (detección)
  3) Fletcher checksum (8/16/32)
Opción: 3
Ingrese el mensaje en binario (p.ej. 110101): 1111111111101
Seleccione Fletcher (8,16,32): 16

[Nota] Para el cálculo se aplicó padding de 3 bit(s) 0 a la IZQUIERDA (agrupación en 8 bits).

[EMISOR:Fletcher-16] Checksum = 0011111100011110
[EMISOR:Fletcher-16] Trama enviada: 1111111111101001111110001111 0 (mensaje + 16 bits de checksum)
PS C:\Users\JM\Documents\Redes\Esquemas-de-deteccio-n-y-correccio-n>
```

```
PS C:\Users\JM\Documents\Redes\Esquemas-de-deteccio-n-y-correccio-n> java .\Receptor.java
Seleccione algoritmo del RECEPTOR:
  1) Hamming
  2) CRC-32
  3) Fletcher checksum
Opción: 3
Ingrese la trama binaria recibida: 01111111111010011111100011111
Seleccione Fletcher (8,16,32): 16
[RECEPTOR:Fletcher-16] Se detectaron errores. Mensaje descartado.
PS C:\Users\JM\Documents\Redes\Esquemas-de-deteccio-n-y-correccio-n>
```

- **¿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, cada uno de los métodos de detección tiene alguna limitación o una forma en la que puede llegar a dar un falso positivo sobre el mensaje que se está enviando.

  - Hamming:

    Este puede corregir un error en un bit o detectar dos errores en dos bits, sin embargo, si se modifican 3 o más bits el algoritmo puede llegar a detectar la cadena como válida y hacer una traducción a otra cadena distinta de la que se le mandó al emisor.

```
Seleccione algoritmo del EMISOR:
  1) Hamming (corrección de 1 bit)
  2) CRC-32 (detección)
  3) Fletcher checksum (8/16/32)
Opción: 1
Ingrese el mensaje en binario (p.ej. 110101): 1011

[EMISOR:Hamming] Trama enviada: 01100110
```

```
Seleccione algoritmo del RECEPTOR:
  1) Hamming
  2) CRC-32
  3) Fletcher checksum
Opción: 1
Ingrese la trama binaria recibida: 01111010
[RECEPTOR:Hamming-SECDED] Se corrigió 1 bit en posición 7. Mensaje corregido: 1100
```

  - CRC-32

    En este caso, este método crea una trama que se interpreta como un

polinomio de 1s y 0s. Lo que hace el receptor es dividir hasta que el resultado sea 0. Esto quiere decir que siempre que la cadena que se mande sea un múltiplo de la cadena original (es decir la transformación de 1) la cadena será aceptada. Para que sea múltiplo se le tiene que hacer un XOR.

```
Seleccione algoritmo del EMISOR:
  1) Hamming (corrección de 1 bit)
  2) CRC-32 (detección)
  3) Fletcher checksum (8/16/32)
Opción: 2
Ingrese el mensaje en binario (p.ej. 110101): 1101

[EMISOR:CRC-32] Trama enviada: 11010011000111001101100001101101 0011 (mensaje + 32 bits de CRC)
```

```
11010011000111001101100001101101 0011
    10000010011000001000111011011 0111
11000011010100001100100110110110 0100
```

```
Seleccione algoritmo del RECEPTOR:
  1) Hamming
  2) CRC-32
  3) Fletcher checksum
Opción: 2
Ingrese la trama binaria recibida: 11000011010100001100100110110110 0100
[RECEPTOR:CRC-32] Sin errores. Mensaje original: 1100
```

- Fletcher

  Este método se basa en sumas modulares. Esto quiere decir qué sí se afecta cada parte de la cadena para que el número "no cambie" a nivel de módulo, el cual depende del tipo de fletcher que se escoja. Por ejemplo, si se usa el fletcher de 8, el mensaje se dividirá cada 4 bits y el módulo será 15 porque es el número más grande que pueden representar 4 bits. Si tomamos una cadena así: 1101 0110 0000, donde el primer bloque representa a 13, el segundo a 6 y el tercero a 0, va a devolver un checksum de 10100101.

```
Seleccione algoritmo del EMISOR:
  1) Hamming (corrección de 1 bit)
  2) CRC-32 (detección)
  3) Fletcher checksum (8/16/32)
Opción: 3
Ingrese el mensaje en binario (p.ej. 110101): 110101100000
Seleccione Fletcher (8,16,32): 8

[EMISOR:Fletcher-8] Checksum = 10100101
[EMISOR:Fletcher-8] Trama enviada: 110101100000 10100101 (mensaje + 8 bits de checksum)
```

  Sin embargo, si se respeta la suma modular, se puede conseguir ese mismo checksum pero con otro patrón de números. Por ejemplo, la cadena 1110 0100 0001, que representan a 14, 4 y 1 respectivamente, sigue teniendo el mismo valor en mod 15 por lo que el checksum es nuevamente 10100101.

```
Seleccione algoritmo del EMISOR:
  1) Hamming (corrección de 1 bit)
  2) CRC-32 (detección)
  3) Fletcher checksum (8/16/32)
Opción: 3
Ingrese el mensaje en binario (p.ej. 110101): 111001000001
Seleccione Fletcher (8,16,32): 8

[EMISOR:Fletcher-8] Checksum = 10100101
[EMISOR:Fletcher-8] Trama enviada: 1110010000110100101 (mensaje + 8 bits de checksum)
```

Por lo tanto, si se cambian los primeros bits pero se mantiene el checksum, el receptor acepta ambos sin problemas.

```
Seleccione algoritmo del RECEPTOR:
  1) Hamming
  2) CRC-32
  3) Fletcher checksum
Opción: 3
Ingrese la trama binaria recibida: 11010110000010100101
Seleccione Fletcher (8,16,32): 8
[RECEPTOR:Fletcher-8] Sin errores. Mensaje original: 110101100000
```

```
Seleccione algoritmo del RECEPTOR:
  1) Hamming
  2) CRC-32
  3) Fletcher checksum
Opción: 3
Ingrese la trama binaria recibida: 11100100000110100101
Seleccione Fletcher (8,16,32): 8
[RECEPTOR:Fletcher-8] Sin errores. Mensaje original: 111001000001
```

Por lo tanto, sí es posible cambiar los bits en las cadenas de forma que cualquiera de los algoritmos sea incapaz de detectarlos. Para el método de corrección Hamming es bastante fácil, ya que solo se requiere un cambio de 3 bits. Sin embargo, para los métodos de detección es más complicado ya que se necesita un cambio muy específico en los bits. Algo que en la vida real es más improbable que ocurra.

- **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 [...]."**

Si tomamos a los 3 algoritmos implementados como solo de detección de errores, podemos realizar un cuadro comparativo sobre su complejidad, velocidad y redundancia. Por ejemplo, podemos decir que CRC y Fletcher se escalan mejor a tramas muy grandes, porque introducen una redundancia constante, mientras hamming agrega Log(M) bits adicionales. Sin embargo, hamming es también un algoritmo de detección de errores lo que le da un uso mayor que los otros. En términos de complejidad, CRC es el único que realiza operaciones polinomiales, mientras fletcher y hamming usan sumas simples o compuertas, lo que con un hardware especializado los puede hacer muy eficientes.

Característica evaluada	**Algoritmo**		
	Hamming	**CRC-32**	**Fletcher (16 principalmente)**
Complejidad	Aunque el algoritmo per se es simple, su implementación y uso varia en el caso, lo que puede llevar a un poco más de complejidad, pero es bastante simple en su idea.	Este es un algoritmo un poco más complejo debido a que introduce operaciones polinomiales en lugar de solo compuertas o sumas.	Tiene una complejidad baja y solo se basa en sumas de modulo y puede ser más simple de implementar que los algoritmos anteriores
Velocidad	Hamming puede llegar a ser muy rápido, pues solo comprende de una operación de logaritmo y compuertas XOR que pueden ser sencillas de calcular incluso a nivel de hardware	CRC es un algoritmo bastante rápido, debido a su uso de diccionarios o estructuras basadas en lookup. Aprovechamos información previamente calculada para llevar a mejor tiempo de ejecución.	Es más rápido en términos de software, pues requiere de menor poder de cómputo que por ejemplo CRC. Aunque, si tomamos en cuenta optimizaciones de hardware, puede quedarse atrás.
redundancia	En términos de redundancia, tiene una muy buena métrica pues para un mensaje de M bits agrega cerca de log(M), el cual en términos de notación asintótica es mucho menor a otros	La redundancia de CRC-32 es constante (32 bits). Lo cual para mensajes cortos puede ser mucho, pero para mensajes en KB o MB es despreciable. Es un algoritmo diseñado para ser escalable y reducir redundancia en mensajes muy grandes.	Nuevamente, su redundancia es fija y depende de su variación. En versiones como la 8 o 16 puede ser más compacto que CRC y por tanto se adapta mejor a tramas pequeñas.

Como conclusión a esta comparación, hamming no solo es un algoritmo de detección, sino que también de corrección para variaciones pequeñas, sin embargo, introduce una redundancia que depende de la longitud de la trama. CRC es el más complejo y no necesariamente el más rápido, pero incluye una redundancia constante. Por último, fletcher es el algoritmo de detección más optimizado de los evaluados, pues introduce poca redundancia y es simple y flexible.

Parte 2

Descripción

En esta segunda parte del laboratorio, se unirán los métodos de corrección/detección de errores implementados anteriormente. Se realizo una comunicación entre estos a partir de la generación de servicios similares a los presentes en la estructura OSI de redes. Precisamente una de aplicación, presentación, enlace y transmisión. A continuación se describe brevemente la implementación de cada una de estas capas para el programa receptor y emisor.

Implementacion

- *Emisor:*
  - **Aplicación:** se permite escoger el algoritmo a utilizar para la detección de errores, así como también permite el ingreso de una cadena de caracteres para manejar mensajes interpretables
  - **Presentación:** para la capa de presentación, se codifica el mensaje escrito por medio de ASCII-UTF8 en una representación donde un byte representa cada carácter. Asimismo, se introduce un pseudo encabezado, donde se asigna un número del 1 al 5 en representación binaria para que el receptor pueda identificar que algoritmos aplicar.
  - **Enlace:** el programa aplica el algoritmo escogido a la cadena inicial convertida en binario y genera una trama lista para ser enviada.
  - **Ruido:** a esta misma trama, se le aplica un nivel parametrizable de ruido. Este ruido consiste en una probabilidad uniforme donde cada bit tiene el mismo chance de cambiar de 1 a 0 o viceversa. Para esta implementación, no se aplicó ruido al encabezado, para asegurar que el receptor siempre use el algoritmo adecuado.
  - **Transmisión:** con el uso de websockets, se plantea un flujo cliente-servidor para los programas emisores y receptores, donde en este se envía la información procesada por las capas anteriores.

- *Receptor:*
    - **Enlace:** en la etapa de enlace, primero se seccionan los primeros 8 bits del contenido recibido y se identifica el algoritmo a aplicar. Como se mencionó anteriormente, podemos confiar en el encabezado, o habrá ruido. Luego de identificar el algoritmo, se aplica la verificación o corrección del contenido según el caso.
    - **Presentación:** en la parte de presentación, se identifica el caso de la corrección y detección de errores, si el mensaje está intacto, si se pudo corregir o si hubo un error (error no corregible por corrección, o simplemente detectado). Asimismo, si el mensaje es interpretable, se aplica una decodificación donde se convierte el contenido binario a una cadena de caracteres.
    - **Aplicación:** por último, se imprimen en consola la trama recibida, el encabezado y el estatus del manejo de errores. En caso el mensaje sea interpretable, entonces se imprime la cadena.

Metodologia

Para poder evaluar el funcionamiento de estos algoritmos además de implementar las capas para uso manual de un usuario, se automatizo para la evaluación de más de 10 mil pruebas. Este proceso consiste en los siguientes pasos, manejados principalmente por el emisor:

- **Iteración:** las pruebas automatizadas comienzan por una iteración que evalúa 1000 envíos de mensajes para cada algoritmo, longitud de mensajes y una incremental probabilidad de ruido.
- **Generacion:** dada la longitud del mensaje y el algoritmo a utilizar, se genera una cadena aleatoria de caracteres, de los cuales se codifica su integridad y se les aplica el ruido a las tramas
- **Transmisión:** luego de ser generado el mensaje completo, se envía a través de la conexión por websockets, donde luego el emisor recibe si el estatus de la integridad del mensaje y su tiempo de ejecución para detectar o corregir errores.
- **Almacenamiento:** por último, en cada una de estas iteraciones, se almacena dentro de un csv las condiciones de cada mensaje enviado para su análisis posterior

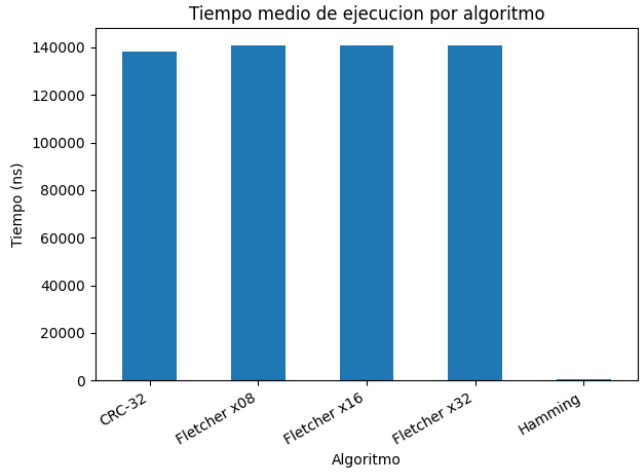
Resultados

- **Tiempo**

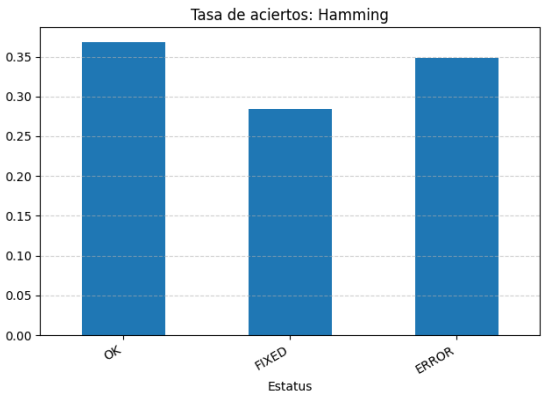
Universidad del Valle de Guatemala
Facultad de Ingeniería
Departamento de Ciencias de la Computación
CC3067 - Redes

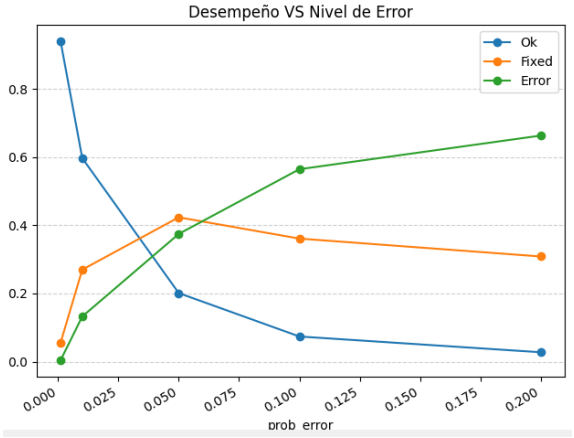
Laboratorio #2
Parte 1 y 2

17/08/2025
Sofía Velásquez, 22049
José Rodrigo Marchena, 22398
Diego García, 22404
Semestre II - 2025

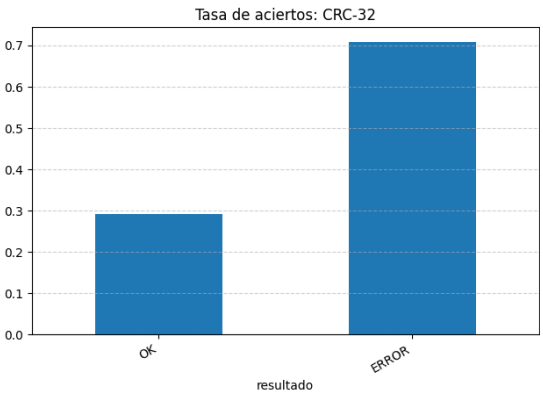


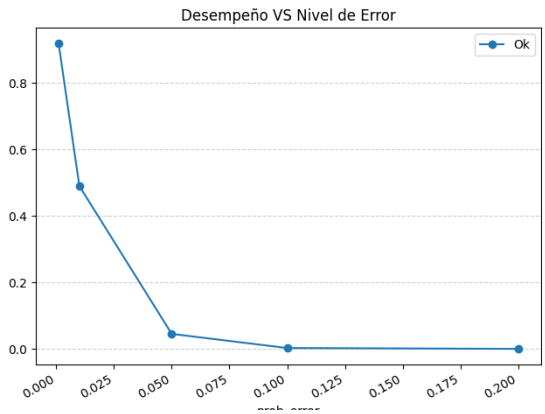
- **Hamming**





- CRC-32



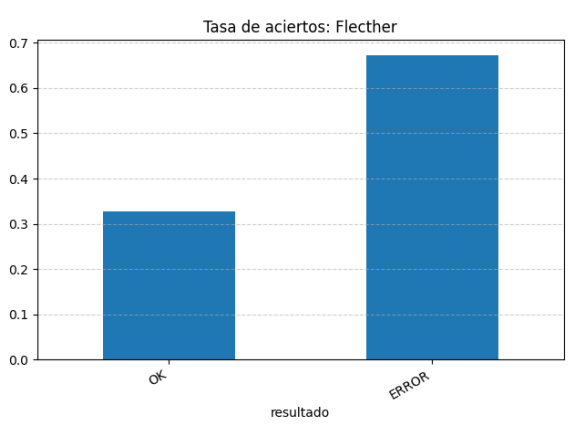


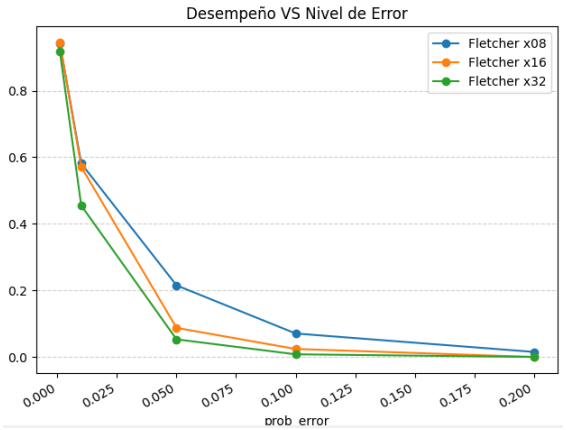
- Fletcher

Universidad del Valle de Guatemala
Facultad de Ingeniería
Departamento de Ciencias de la Computación
CC3067 - Redes

Laboratorio #2
Parte 1 y 2

17/08/2025
Sofía Velásquez, 22049
José Rodrigo Marchena, 22398
Diego García, 22404
Semestre II - 2025





Discusión

Los resultados de hamming muestran no solo la tasa de mensajes intactos, sino también aquellos que pudieron ser arreglados y los que no. Se puede observar en la distribución de barras que, a pesar de aumentar considerablemente el nivel de error, el algoritmo de hamming tiene básicamente un 66% de probabilidad de poder interpretar correctamente un mensaje. Asimismo, en la distribución comparando el nivel de error, se puede observar que previo a un porcentaje de error de 0.05%, la cantidad de mensajes interpretables supera aquellos con errores fatales. Asimismo, luego de 0.1% de ruido, los errores comienzan a volverse mayoría, lo cual nos puede dar un indicio de en qué circunstancias este algoritmo será apropiado.

Para los algoritmos de detección, CRC y Hamming tuvieron un nivel de aciertos cercano al 30% de los mensajes recibidos. El nivel de diferencia entre estos podría deberse a ruido de la misma distribución uniforme. Asimismo, es posible observar que incluso con aumentos ligeros en la probabilidad, la cantidad de menajes intactos disminuye bastante.

Conclusiones

- El algoritmo de hamming de errores halla una gran utilidad en la corrección de errores, incluso al aumentar considerablemente la cantidad de ruido. Esto se atribuye al uso también de un bit de paridad completa encima de la estructura original de este algoritmo.
- Asimismo, hamming mostro ser el algoritmo con menor tiempo de ejecución en las implementaciones realizadas
- CRC-32 y Fletcher mostraron tener un rendimiento comparable incluso al variar el tamaño de las tramas.

Referencias

How is a CRC32 checksum calculated? (n.d.). Stack Overflow. https://stackoverflow.com/questions/2587766/how-is-a-crc32-checksum-calculated

Kapoor, A. (2024, November 17). *What is hamming code, the network technique to detect errors and correct data*. Simplilearn.com. https://www.simplilearn.com/tutorials/networking-tutorial/what-is-hamming-code-technique-to-detect-errors-correct-data

Nakassis, A. (1988). Fletcher's error detection algorithm: how to implement it efficiently and how toavoid the most common pitfalls. *ACM SIGCOMM Computer Communication Review*, *18*(5), 63–88. https://doi.org/10.1145/53644.53648