UNIVERSIDAD DEL VALLE DE GUATEMALA

Facultad de Ingeniería

Laboratorio 2

Esquemas de Detección y Corrección

Edwin Andrés Ortega Kou – 22305

Esteban Zambrano Garoz – 22119

Redes

Guatemala, agosto 2025

Algoritmos seleccionados

Hamming (SEC):

- Propósito, cálculo de bits de paridad, fórmula $2^r \geq m + r + 1$.
- Capacidad: corrige 1 bit, detecta 2 pero no siempre.

CRC-32:

- Concepto de polinomio generador.
- Características: detección de 1 bit, mayoría de dobles, ráfagas ≤ 32 bits.
- No corrige, solo detecta.

Lenguajes utilizados, C++ para emisores y Python para receptores.

Implementación de cada algoritmo

Hamming:

Emitter.cpp

- Entrada: Una cadena binaria (payload) ingresada por teclado en formato MSB a SB.
- Proceso:
    - Calcula la cantidad de bits de paridad r usando la fórmula $2^r \geq m + r + 1$.
    - Inserta los bits de datos en las posiciones que no son potencias de 2.
    - Calcula cada bit de paridad recorriendo las posiciones que cubre y aplicando paridad par.
    - Construye el codeword final en formato MSB a LSB.
- Salida: Muestra en consola el codeword generado listo para ser enviado al receptor.

Receiver.py

- Entrada: El codeword recibido (cadena binaria MSB a LSB).
- Proceso:
    - Convierte la cadena a un vector indexado desde la derecha (LSB = posición 1).
    - Calcula el síndrome recomputando las paridades.
    - Si el síndrome es cero, no hay errores; si es distinto de cero a invierte el bit en esa posición para corregirlo.
    - Extrae el payload original ignorando las posiciones de paridad.
- Salida: Indica si hubo errores, si se corrigieron, y muestra el payload recuperado.

CRC-32:

Emitter.cpp

- Entrada: Una cadena binaria (payload) ingresada por teclado en formato MSB a LSB.
- Proceso:
    - Inicializa el registro del CRC en 0xFFFFFFFF.
    - Recorre cada bit del payload y aplica el algoritmo bit a bit MSB-first con el polinomio generador 0x04C11DB7.
    - Al final, invierte el resultado con xorout = 0xFFFFFFFF.
    - Convierte el CRC a 32 bits binarios y lo concatena al payload.
- Salida: Muestra en consola el CRC calculado y la trama completa (payload + CRC).

Receiver.py

- Entrada: La trama recibida (payload + CRC) en formato binario MSB a LSB.
- Proceso:
    - Separa los últimos 32 bits como CRC recibido y el resto como payload.
    - Calcula nuevamente el CRC del payload recibido usando el mismo polinomio y parámetros que el emisor.
    - Compara el CRC calculado con el CRC recibido.
- Salida: Si son iguales entonces “No se detectaron errores” y muestra el payload; si no “Se detectaron errores” y descarta la trama.

Ejemplos únicamente de los algoritmos

Hamming

Emisor

```
andyok@KouTechSys:~/Redes/Deteccion-Correccion_1/hamming$ ./hamming_emit
Hamming (emisor C++). Ingrese mensaje binario (MSB->LSB): 110101
Codeword: 1100101110
```

```
andyok@KouTechSys:~/Redes/Deteccion-Correccion_1/hamming$ ./hamming_emit
Hamming (emisor C++). Ingrese mensaje binario (MSB->LSB): 1001
Codeword: 1001100
```

```
andyok@KouTechSys:~/Redes/Deteccion-Correccion_1/hamming$ ./hamming_emit
Hamming (emisor C++). Ingrese mensaje binario (MSB->LSB): 01011001
Codeword: 010101001110
```

Receptor

1. Sin errores

110101

```
andyok@KouTechSys:~/Redes/Deteccion-Correccion_1/hamming$ python3 receiver.py
Hamming (receptor Python). Ingrese codeword (MSB->LSB): 1100101110
No se detectaron errores.
Payload: 110101
```

1001

```
andyok@KouTechSys:~/Redes/Deteccion-Correccion_1/hamming$ python3 receiver.py
Hamming (receptor Python). Ingrese codeword (MSB->LSB): 1001100
No se detectaron errores.
Payload: 1001
```

01011001

```
andyok@KouTechSys:~/Redes/Deteccion-Correccion_1/hamming$ python3 receiver.py
Hamming (receptor Python). Ingrese codeword (MSB->LSB): 010101001110
No se detectaron errores.
Payload: 01011001
```

2. 1 error

110101

```
andyok@KouTechSys:~/Redes/Deteccion-Correccion_1/hamming$ python3 receiver.py
Hamming (receptor Python). Ingrese codeword (MSB->LSB): 1100101111
Se detectaron y corrigieron errores.
Bit corregido en posicion 1.
Payload: 110101
```

1001

```
andyok@KouTechSys:~/Redes/Deteccion-Correccion_1/hamming$ python3 receiver.py
Hamming (receptor Python). Ingrese codeword (MSB->LSB): 1000100
Se detectaron y corrigieron errores.
Bit corregido en posicion 4.
Payload: 1001
```

01011001

```
andyok@KouTechSys:~/Redes/Deteccion-Correccion_1/hamming$ python3 receiver.py
Hamming (receptor Python). Ingrese codeword (MSB->LSB): 010001001110
Se detectaron y corrigieron errores.
Bit corregido en posicion 9.
Payload: 01011001
```

3. 2 errores

110101

```
andyok@KouTechSys:~/Redes/Deteccion-Correccion_1/hamming$ python3 receiver.py
Hamming (receptor Python). Ingrese codeword (MSB->LSB): 1100001111
Se detectaron y corrigieron errores.
Bit corregido en posicion 7.
Payload: 111001
```

Esto tiene 2 bits cambiados respecto al original. El receptor dice “se corrigió en posición 7” pero el payload quedará distinto a 110101.

1001

```
andyok@KouTechSys:~/Redes/Deteccion-Correccion_1/hamming$ python3 receiver.py
Hamming (receptor Python). Ingrese codeword (MSB->LSB): 1100100
Se detectaron y corrigieron errores.
Bit corregido en posicion 2.
Payload: 1101
```

01011001

```
andyok@KouTechSys:~/Redes/Deteccion-Correccion_1/hamming$ python3 receiver.py
Hamming (receptor Python). Ingrese codeword (MSB->LSB): 010001001010
Se detectaron y corrigieron errores.
Bit corregido en posicion 10.
Payload: 01101000
```

CRC-32

Emisor

```
andyok@KouTechSys:~/Redes/Deteccion-Correccion_1/crc32$ ./crc_emit
CRC-32 (emisor C++). Ingrese payload binario (MSB->LSB): 110101
CRC32: 00101111100010101101011011101001
Trama||CRC32: 11010100101111100010101101011011101001
```

```
andyok@KouTechSys:~/Redes/Deteccion-Correccion_1/crc32$ ./crc_emit
CRC-32 (emisor C++). Ingrese payload binario (MSB->LSB): 1001
CRC32: 00011010100001100100110110111101
Trama||CRC32: 10010001101010000110010011011101
```

```
andyok@KouTechSys:~/Redes/Deteccion-Correccion_1/crc32$ ./crc_emit
CRC-32 (emisor C++). Ingrese payload binario (MSB->LSB): 01011001
CRC32: 11101011101010010001101101000011
Trama||CRC32: 01011001111010111010100100011011010000011
```

Receptor

1. Sin errores
   110101

```
andyok@KouTechSys:~/Redes/Deteccion-Correccion_1/crc32$ python3 receiver.py
CRC-32 (receptor Python). Ingrese trama||CRC32 (MSB->LSB): 11010100101111100010101101011011101001
No se detectaron errores.
Payload: 110101
```

1001

```
andyok@KouTechSys:~/Redes/Deteccion-Correccion_1/crc32$ python3 receiver.py
CRC-32 (receptor Python). Ingrese trama||CRC32 (MSB->LSB): 10010001101010000110010011011101101
No se detectaron errores.
Payload: 1001
```

01011001

```
andyok@KouTechSys:~/Redes/Deteccion-Correccion_1/crc32$ python3 receiver.py
CRC-32 (receptor Python). Ingrese trama||CRC32 (MSB->LSB): 01011001111010111010100100011011010000011
No se detectaron errores.
Payload: 01011001
```

2. Error en un bit

   110101

```
andyok@KouTechSys:~/Redes/Deteccion-Correccion_1/crc32$ python3 receiver.py
CRC-32 (receptor Python). Ingrese trama||CRC32 (MSB->LSB): 11010100101111100010101101101110100 0
Se detectaron errores.
Trama descartada.
```

   1001

```
andyok@KouTechSys:~/Redes/Deteccion-Correccion_1/crc32$ python3 receiver.py
CRC-32 (receptor Python). Ingrese trama||CRC32 (MSB->LSB): 10010001101010010110010011011011110 1
Se detectaron errores.
Trama descartada.
```

   01011001

```
andyok@KouTechSys:~/Redes/Deteccion-Correccion_1/crc32$ python3 receiver.py
CRC-32 (receptor Python). Ingrese trama||CRC32 (MSB->LSB): 01011001111010101010100100011011010000 11
Se detectaron errores.
Trama descartada.
```

3. Error en múltiples bits

   110101

```
andyok@KouTechSys:~/Redes/Deteccion-Correccion_1/crc32$ python3 receiver.py
CRC-32 (receptor Python). Ingrese trama||CRC32 (MSB->LSB): 11110100101111100010101101101110000 0
Se detectaron errores.
Trama descartada.
```

   1001

```
andyok@KouTechSys:~/Redes/Deteccion-Correccion_1/crc32$ python3 receiver.py
CRC-32 (receptor Python). Ingrese trama||CRC32 (MSB->LSB): 10010001101010010110010011011011010 1
Se detectaron errores.
Trama descartada.
```

   01011001

```
andyok@KouTechSys:~/Redes/Deteccion-Correccion_1/crc32$ python3 receiver.py
CRC-32 (receptor Python). Ingrese trama||CRC32 (MSB->LSB): 01011001111010101010100100011011010100 11
Se detectaron errores.
Trama descartada.
```

¿Es posible manipular los bits para que no se detecte el error?

Hamming:

Si es posible, ya que el código de hamming está diseñado para detectar y corregir un único error por palabra de código. Si se producen dos o más errores en posiciones distintas, existe la posibilidad de que se “corrija” mal un bit que no está equivocado, alterando el mensaje; que el patrón de error haga que el síndrome sea 0, por lo que el error pasa desapercibido.

CRC-32:

Teóricamente sí, pero en la práctica es muy poco improbable para datos cortos. Un error pasa inadvertido si el patrón de bits alterados produce el mismo resto al dividir el mensaje por el

polinomio generador de CRC-32. Para lograrlo de forma intencional, el atacante necesitaría conocer el polinomio y calcular un patrón específico de errores.

Ventajas y desventajas de cada algoritmo

Hamming

Ventajas:

- Capaz de corregir un error de un solo bit sin retransmisión.
- Detecta errores de un bit y algunos errores múltiples.
- Overhead bajo: requiere log2(n+1) bits de paridad para una palabra de n bits.

Desventajas:

- No corrige errores múltiples y, en algunos casos, los interpreta mal (error no detectado o corrección equivocada).
- Más complejo que un simple bit de paridad, requiere cálculo de posiciones y operaciones XOR adicionales.

CRC-32

Ventajas:

- Muy alta capacidad de detección de errores, incluyendo ráfagas de hasta 32 bits.
- Bajo riesgo de colisiones en mensajes cortos o medianos.
- Rápido de calcular usando operaciones binarias.

Desventajas:

- No corrige errores; solo detecta.
- Overhead fijo de 32 bits, que puede ser significativo en mensajes muy pequeños.
- Implementación ligeramente más compleja que un bit de paridad.

Resumen del programa

Se implementó una arquitectura en capas (Aplicación, Presentación, Enlace, Ruido y Transmisión) para evaluar dos esquemas del nivel de enlace: Hamming (corrección de un bit, SEC) y CRC-32 (detección). El emisor está en C++ y el receptor en Python, comunicados por sockets TCP y un canal con errores tipo BSC (Binary Symmetric Channel) con probabilidad de volteo por bit p. Se automatizaron pruebas masivas más de 10mil mensajes por punto, variando el tamaño del mensaje siendo estos 4, 8, 16 y 32 bytes, también p donde toma valores de 0, 0.005, 0.01, 0.02 y 0.05. A partir de estas simulaciones se generaron CSV's y en base a esos se calcularon métricas de entrega correcta (OK+CORRECTED)/N y descartes DISC/N, por último se graficó su comportamiento.

En términos generales, Hamming mantiene alta entrega correcta para *p* bajos al corregir un error, mientras que CRC-32 ofrece detección muy confiable a costa de mayores descartes cuando *p* crece.

Metodología experimental del programa

Esta sección describe exactamente cómo se generaron los datos, qué parámetros se variaron, cómo se midieron los resultados y cómo se produjeron las gráficas.

Factores de variación

- Algoritmo: HAM y CRC.
- Longitud del mensaje: 4, 8, 16, 32 bytes.
- Probabilidad de error del canal BSC: p = {0.00, 0.005, 0.01, 0.02, 0.05}.
- Muestras por punto: 10,000 mensajes aleatorios.

Procedimiento del programa

1. Se levanta el receptor (Python) con server.py en una terminal.

```
esteb@Desktop-EZ:~/4to/segundo/redes/Deteccion-Correccion_1/receiver_py$ python3 server.py
[TRANSMISIÓN][RECEPTOR] escuchando en 0.0.0.0:5050
```

2. Se levanta el emisor (C++), que solicita al usuario el mensaje de texto, el algoritmo a usar (HAM o CRC32) y la probabilidad de ruido.

```
esteb@Desktop-EZ:~/4to/segundo/redes/Deteccion-Correccion_1/emitter_cpp$ make
mkdir -p bin
g++ -std=c++17 -O2 -Wall -o bin/emitter src/main.cpp src/application.cpp src/presentation.cpp src/enlace_hamming.cpp src
/enlace_crc32.cpp src/noise.cpp src/net_client.cpp
esteb@Desktop-EZ:~/4to/segundo/redes/Deteccion-Correccion_1/emitter_cpp$ ./bin/emitter
Mensaje (texto): Hola
Algoritmo [HAM|CRC32]: HAM
Probabilidad de ruido p (ej 0.01): 0.01
```

3. El emisor convierte el mensaje a binario ASCII, le añade los bits de control según el algoritmo elegido y muestra en pantalla la trama generada.

```
esteb@Desktop-EZ:~/4to/segundo/redes/Deteccion-Correccion_1/emitter_cpp$ ./bin/emitter
Mensaje (texto): Hola
Algoritmo [HAM|CRC32]: HAM
Probabilidad de ruido p (ej 0.01): 0.01
[APLICACIÓN][EMISOR] Mensaje: "Hola", Algoritmo=HAM, p_ruido=0.01
[PRESENTACIÓN][EMISOR] ASCII->bits: 01001000011011110110110001100001
```

4. Se aplica el ruido Bernoulli(p), mostrando las posiciones alteradas y la trama final con ruido.

```
[ENLACE][EMISOR][HAM] layout (MSB->LSB): d d d d d d p d d d d d d d d d d d d d p d d d d d d d p d d d p d p p
[ENLACE][EMISOR][HAM] m=32 bits, r=6 paridades, codeword: 010010000011011110110111000110100000111
[RUIDO][EMISOR] p=0.01, posiciones volteadas (MSB=1): NINGUNA
[RUIDO][EMISOR] Trama con ruido: 010010000011011110110111000110100000111
```

5. El receptor procesa la trama:
    a. HAM: calcula el síndrome, corrige si hay error y devuelve el mensaje.
    b. CRC32: compara el CRC recibido y recalculado; si no coinciden descarta el mensaje.

```
esteb@Desktop-EZ:~/4to/segundo/redes/Deteccion-Correccion_1/receiver_py$ python3 server.py
[TRANSMISIÓN][RECEPTOR] escuchando en 0.0.0.0:5050
[TRANSMISIÓN][RECEPTOR] conexión de ('127.0.0.1', 58010)
[TRANSMISIÓN][RECEPTOR] JSON recibido: {"alg":"HAM","frame_bits":"0100100000110111101101110001101000011 1"}
[APLICACIÓN][RECEPTOR] Algoritmo recibido: HAM
[ENLACE][RECEPTOR][HAM] codeword recibido (MSB->LSB): 01 0010 0000 1101 1110 1101 1100 0110 1000 0111
[ENLACE][RECEPTOR][HAM] layout (MSB->LSB):            d d d d d d p d d d d d d d d d d d d d p d d d d d d d d p d d
d p d p p
[ENLACE][RECEPTOR][HAM] Síndrome = 0 → OK (sin errores).
[ENLACE][RECEPTOR][HAM] Payload (MSB->LSB):          0100 1000 0110 1111 0110 1100 0110 0001
[PRESENTACIÓN][RECEPTOR] bits->ASCII: "Hola"
[APLICACIÓN][RECEPTOR] Mostrar mensaje: "Hola" (status=OK)
[TRANSMISIÓN][RECEPTOR] Respuesta: {"status": "OK", "info": {"syndrome": 0}, "decoded_text": "Hola"}
```

6. Finalmente, el receptor responde al emisor con un JSON indicando el estado (OK, CORRECTED o DISCARD) y, si es válido, el mensaje decodificado.

```
Respuesta receptor: {"status": "OK", "info": {"syndrome": 0}, "decoded_text": "Hola"}
```

Procedimiento de pruebas

1. Se levanta el receptor (Python) con server.py en una terminal.

```
esteb@Desktop-EZ:~/4to/segundo/redes/Deteccion-Correccion_1/receiver_py$ python3 server.py
[TRANSMISIÓN][RECEPTOR] escuchando en 0.0.0.0:5050
```

2. Se ejecuta el cliente haciendo uso de comando los cuales generan mensajes que codifican en HAM o CRC32, aplica ruido Bernoulli(p), envía por socket y guarda los resultados en la carpeta results/*.csv.

```
esteb@Desktop-EZ:~/4to/segundo/redes/Deteccion-Correccion_1/experiments$ python3 run_batch.py --alg HAM --p 0.00  --n 10
000 --len 4  --csv results/ham_p0.00_len4.csv
python3 run_batch.py --alg HAM --p 0.005 --n 10000 --len 4  --csv results/ham_p0.005_len4.csv
python3 run_batch.py --alg HAM --p 0.01  --n 10000 --len 4  --csv results/ham_p0.01_len4.csv
python3 run_batch.py --alg HAM --p 0.02  --n 10000 --len 4  --csv results/ham_p0.02_len4.csv
python3 run_batch.py --alg HAM --p 0.05  --n 10000 --len 4  --csv results/ham_p0.05_len4.csv
[HAM] p=0.0  1000/10000  OK=1000 CORR=0 DISC=0
[HAM] p=0.0  2000/10000  OK=2000 CORR=0 DISC=0
[HAM] p=0.0  3000/10000  OK=3000 CORR=0 DISC=0
[HAM] p=0.0  4000/10000  OK=4000 CORR=0 DISC=0
[HAM] p=0.0  5000/10000  OK=5000 CORR=0 DISC=0
[HAM] p=0.0  6000/10000  OK=6000 CORR=0 DISC=0
[HAM] p=0.0  7000/10000  OK=7000 CORR=0 DISC=0
[HAM] p=0.0  8000/10000  OK=8000 CORR=0 DISC=0
[HAM] p=0.0  9000/10000  OK=9000 CORR=0 DISC=0
[HAM] p=0.0  10000/10000  OK=10000 CORR=0 DISC=0
[FIN] CSV -> results/ham_p0.00_len4.csv  Totales: OK=10000 CORR=0 DISC=0
[HAM] p=0.005  1000/10000  OK=812 CORR=182 DISC=6
[HAM] p=0.005  2000/10000  OK=1644 CORR=344 DISC=12
[HAM] p=0.005  3000/10000  OK=2466 CORR=514 DISC=20
[HAM] p=0.005  4000/10000  OK=3305 CORR=668 DISC=27
[HAM] p=0.005  5000/10000  OK=4129 CORR=842 DISC=29
[HAM] p=0.005  6000/10000  OK=4965 CORR=1002 DISC=33
[HAM] p=0.005  7000/10000  OK=5785 CORR=1178 DISC=37
[HAM] p=0.005  8000/10000  OK=6611 CORR=1345 DISC=44
[HAM] p=0.005  9000/10000  OK=7437 CORR=1516 DISC=47
[HAM] p=0.005  10000/10000  OK=8248 CORR=1702 DISC=50
[FIN] CSV -> results/ham_p0.005_len4.csv  Totales: OK=8248 CORR=1702 DISC=50
[HAM] p=0.01  1000/10000  OK=650 CORR=338 DISC=12
[HAM] p=0.01  2000/10000  OK=1339 CORR=629 DISC=32
[HAM] p=0.01  3000/10000  OK=2037 CORR=911 DISC=52
[HAM] p=0.01  4000/10000  OK=2723 CORR=1212 DISC=65
[HAM] p=0.01  5000/10000  OK=3388 CORR=1530 DISC=82
[HAM] p=0.01  6000/10000  OK=4064 CORR=1846 DISC=90
[HAM] p=0.01  7000/10000  OK=4728 CORR=2165 DISC=107
[HAM] p=0.01  8000/10000  OK=5408 CORR=2474 DISC=118
[HAM] p=0.01  9000/10000  OK=6072 CORR=2802 DISC=126
[HAM] p=0.01  10000/10000  OK=6745 CORR=3121 DISC=134
[FIN] CSV -> results/ham_p0.01_len4.csv  Totales: OK=6745 CORR=3121 DISC=134
```

```
esteb@Desktop-EZ:~/4to/segundo/redes/Deteccion-Correccion_1/experiments$ python3 run_batch.py --alg HAM --p 0.00  --n 10
000 --len 8  --csv results/ham_p0.00_len8.csv
python3 run_batch.py --alg HAM --p 0.005 --n 10000 --len 8  --csv results/ham_p0.005_len8.csv
python3 run_batch.py --alg HAM --p 0.01  --n 10000 --len 8  --csv results/ham_p0.01_len8.csv
python3 run_batch.py --alg HAM --p 0.02  --n 10000 --len 8  --csv results/ham_p0.02_len8.csv
python3 run_batch.py --alg HAM --p 0.05  --n 10000 --len 8  --csv results/ham_p0.05_len8.csv
[HAM] p=0.0  1000/10000  OK=1000 CORR=0 DISC=0
[HAM] p=0.0  2000/10000  OK=2000 CORR=0 DISC=0
[HAM] p=0.0  3000/10000  OK=3000 CORR=0 DISC=0
[HAM] p=0.0  4000/10000  OK=4000 CORR=0 DISC=0
[HAM] p=0.0  5000/10000  OK=5000 CORR=0 DISC=0
[HAM] p=0.0  6000/10000  OK=6000 CORR=0 DISC=0
[HAM] p=0.0  7000/10000  OK=7000 CORR=0 DISC=0
[HAM] p=0.0  8000/10000  OK=8000 CORR=0 DISC=0
[HAM] p=0.0  9000/10000  OK=9000 CORR=0 DISC=0
[HAM] p=0.0  10000/10000  OK=10000 CORR=0 DISC=0
[FIN] CSV -> results/ham_p0.00_len8.csv  Totales: OK=10000 CORR=0 DISC=0
```

```
[HAM] p=0.05  1000/10000  OK=41 CORR=723 DISC=236
[HAM] p=0.05  2000/10000  OK=78 CORR=1453 DISC=469
[HAM] p=0.05  3000/10000  OK=112 CORR=2194 DISC=694
[HAM] p=0.05  4000/10000  OK=138 CORR=2923 DISC=939
[HAM] p=0.05  5000/10000  OK=172 CORR=3644 DISC=1184
[HAM] p=0.05  6000/10000  OK=202 CORR=4372 DISC=1426
[HAM] p=0.05  7000/10000  OK=230 CORR=5090 DISC=1680
[HAM] p=0.05  8000/10000  OK=263 CORR=5826 DISC=1911
[HAM] p=0.05  9000/10000  OK=294 CORR=6548 DISC=2158
[HAM] p=0.05  10000/10000  OK=330 CORR=7283 DISC=2387
[FIN] CSV -> results/ham_p0.05_len8.csv  Totales: OK=330 CORR=7283 DISC=2387
esteb@Desktop-EZ:~/4to/segundo/redes/Deteccion-Correccion_1/experiments$
```

```
esteb@Desktop-EZ:~/4to/segundo/redes/Deteccion-Correccion_1/experiments$ python3 run_batch.py --alg HAM --p 0.00  --n 10
000 --len 16 --csv results/ham_p0.00_len16.csv
python3 run_batch.py --alg HAM --p 0.005 --n 10000 --len 16 --csv results/ham_p0.005_len16.csv
python3 run_batch.py --alg HAM --p 0.01  --n 10000 --len 16 --csv results/ham_p0.01_len16.csv
python3 run_batch.py --alg HAM --p 0.02  --n 10000 --len 16 --csv results/ham_p0.02_len16.csv
python3 run_batch.py --alg HAM --p 0.05  --n 10000 --len 16 --csv results/ham_p0.05_len16.csv
[HAM] p=0.0  1000/10000  OK=1000 CORR=0 DISC=0
[HAM] p=0.0  2000/10000  OK=2000 CORR=0 DISC=0
[HAM] p=0.0  3000/10000  OK=3000 CORR=0 DISC=0
[HAM] p=0.0  4000/10000  OK=4000 CORR=0 DISC=0
[HAM] p=0.0  5000/10000  OK=5000 CORR=0 DISC=0
[HAM] p=0.0  6000/10000  OK=6000 CORR=0 DISC=0
[HAM] p=0.0  7000/10000  OK=7000 CORR=0 DISC=0
[HAM] p=0.0  8000/10000  OK=8000 CORR=0 DISC=0
[HAM] p=0.0  9000/10000  OK=9000 CORR=0 DISC=0
[HAM] p=0.0  10000/10000  OK=10000 CORR=0 DISC=0
[FIN] CSV -> results/ham_p0.00_len16.csv  Totales: OK=10000 CORR=0 DISC=0
```

```
[HAM] p=0.05  1000/10000  OK=6 CORR=671 DISC=323
[HAM] p=0.05  2000/10000  OK=8 CORR=1389 DISC=603
[HAM] p=0.05  3000/10000  OK=12 CORR=2094 DISC=894
[HAM] p=0.05  4000/10000  OK=19 CORR=2797 DISC=1184
[HAM] p=0.05  5000/10000  OK=25 CORR=3507 DISC=1468
[HAM] p=0.05  6000/10000  OK=32 CORR=4193 DISC=1775
[HAM] p=0.05  7000/10000  OK=40 CORR=4927 DISC=2033
[HAM] p=0.05  8000/10000  OK=42 CORR=5658 DISC=2300
[HAM] p=0.05  9000/10000  OK=50 CORR=6383 DISC=2567
[HAM] p=0.05  10000/10000  OK=57 CORR=7105 DISC=2838
[FIN] CSV -> results/ham_p0.05_len16.csv  Totales: OK=57 CORR=7105 DISC=2838
esteb@Desktop-EZ:~/4to/segundo/redes/Deteccion-Correccion_1/experiments$
```

```
esteb@Desktop-EZ:~/4to/segundo/redes/Deteccion-Correccion_1/experiments$ python3 run_batch.py --alg HAM --p 0.00  --n 10
000 --len 32 --csv results/ham_p0.00_len32.csv
python3 run_batch.py --alg HAM --p 0.005 --n 10000 --len 32 --csv results/ham_p0.005_len32.csv
python3 run_batch.py --alg HAM --p 0.01  --n 10000 --len 32 --csv results/ham_p0.01_len32.csv
python3 run_batch.py --alg HAM --p 0.02  --n 10000 --len 32 --csv results/ham_p0.02_len32.csv
python3 run_batch.py --alg HAM --p 0.05  --n 10000 --len 32 --csv results/ham_p0.05_len32.csv
[HAM] p=0.0  1000/10000  OK=1000 CORR=0 DISC=0
[HAM] p=0.0  2000/10000  OK=2000 CORR=0 DISC=0
[HAM] p=0.0  3000/10000  OK=3000 CORR=0 DISC=0
[HAM] p=0.0  4000/10000  OK=4000 CORR=0 DISC=0
[HAM] p=0.0  5000/10000  OK=5000 CORR=0 DISC=0
[HAM] p=0.0  6000/10000  OK=6000 CORR=0 DISC=0
[HAM] p=0.0  7000/10000  OK=7000 CORR=0 DISC=0
[HAM] p=0.0  8000/10000  OK=8000 CORR=0 DISC=0
[HAM] p=0.0  9000/10000  OK=9000 CORR=0 DISC=0
[HAM] p=0.0  10000/10000  OK=10000 CORR=0 DISC=0
[FIN] CSV -> results/ham_p0.00_len32.csv  Totales: OK=10000 CORR=0 DISC=0
```

```
[HAM] p=0.05  1000/10000  OK=3 CORR=674 DISC=323
[HAM] p=0.05  2000/10000  OK=3 CORR=1342 DISC=655
[HAM] p=0.05  3000/10000  OK=3 CORR=2019 DISC=978
[HAM] p=0.05  4000/10000  OK=5 CORR=2712 DISC=1283
[HAM] p=0.05  5000/10000  OK=8 CORR=3397 DISC=1595
[HAM] p=0.05  6000/10000  OK=10 CORR=4074 DISC=1916
[HAM] p=0.05  7000/10000  OK=14 CORR=4761 DISC=2225
[HAM] p=0.05  8000/10000  OK=16 CORR=5479 DISC=2505
[HAM] p=0.05  9000/10000  OK=17 CORR=6169 DISC=2814
[HAM] p=0.05  10000/10000  OK=20 CORR=6841 DISC=3139
[FIN] CSV -> results/ham_p0.05_len32.csv  Totales: OK=20 CORR=6841 DISC=3139
esteb@Desktop-EZ:~/4to/segundo/redes/Deteccion-Correccion_1/experiments$
```

```
esteb@Desktop-EZ:~/4to/segundo/redes/Deteccion-Correccion_1/experiments$ python3 run_batch.py --alg CRC --p 0.00  --n 10
000 --len 4  --csv results/crc_p0.00_len4.csv
python3 run_batch.py --alg CRC --p 0.005 --n 10000 --len 4  --csv results/crc_p0.005_len4.csv
python3 run_batch.py --alg CRC --p 0.01  --n 10000 --len 4  --csv results/crc_p0.01_len4.csv
python3 run_batch.py --alg CRC --p 0.02  --n 10000 --len 4  --csv results/crc_p0.02_len4.csv
python3 run_batch.py --alg CRC --p 0.05  --n 10000 --len 4  --csv results/crc_p0.05_len4.csv
[CRC] p=0.0  1000/10000  OK=1000 CORR=0 DISC=0
[CRC] p=0.0  2000/10000  OK=2000 CORR=0 DISC=0
[CRC] p=0.0  3000/10000  OK=3000 CORR=0 DISC=0
[CRC] p=0.0  4000/10000  OK=4000 CORR=0 DISC=0
[CRC] p=0.0  5000/10000  OK=5000 CORR=0 DISC=0
[CRC] p=0.0  6000/10000  OK=6000 CORR=0 DISC=0
[CRC] p=0.0  7000/10000  OK=7000 CORR=0 DISC=0
[CRC] p=0.0  8000/10000  OK=8000 CORR=0 DISC=0
[CRC] p=0.0  9000/10000  OK=9000 CORR=0 DISC=0
[CRC] p=0.0  10000/10000  OK=10000 CORR=0 DISC=0
[FIN] CSV -> results/crc_p0.00_len4.csv  Totales: OK=10000 CORR=0 DISC=0
```

```
[CRC] p=0.05  1000/10000  OK=38 CORR=0 DISC=962
[CRC] p=0.05  2000/10000  OK=86 CORR=0 DISC=1914
[CRC] p=0.05  3000/10000  OK=124 CORR=0 DISC=2876
[CRC] p=0.05  4000/10000  OK=165 CORR=0 DISC=3835
[CRC] p=0.05  5000/10000  OK=200 CORR=0 DISC=4800
[CRC] p=0.05  6000/10000  OK=244 CORR=0 DISC=5756
[CRC] p=0.05  7000/10000  OK=278 CORR=0 DISC=6722
[CRC] p=0.05  8000/10000  OK=309 CORR=0 DISC=7691
[CRC] p=0.05  9000/10000  OK=345 CORR=0 DISC=8655
[CRC] p=0.05  10000/10000  OK=380 CORR=0 DISC=9620
[FIN] CSV -> results/crc_p0.05_len4.csv  Totales: OK=380 CORR=0 DISC=9620
esteb@Desktop-EZ:~/4to/segundo/redes/Deteccion-Correccion_1/experiments$
```

```
esteb@Desktop-EZ:~/4to/segundo/redes/Deteccion-Correccion_1/experiments$ python3 run_batch.py --alg CRC --p 0.00  --n 10
000 --len 8  --csv results/crc_p0.00_len8.csv
python3 run_batch.py --alg CRC --p 0.005 --n 10000 --len 8  --csv results/crc_p0.005_len8.csv
python3 run_batch.py --alg CRC --p 0.01  --n 10000 --len 8  --csv results/crc_p0.01_len8.csv
python3 run_batch.py --alg CRC --p 0.02  --n 10000 --len 8  --csv results/crc_p0.02_len8.csv
python3 run_batch.py --alg CRC --p 0.05  --n 10000 --len 8  --csv results/crc_p0.05_len8.csv
[CRC] p=0.0  1000/10000  OK=1000 CORR=0 DISC=0
[CRC] p=0.0  2000/10000  OK=2000 CORR=0 DISC=0
[CRC] p=0.0  3000/10000  OK=3000 CORR=0 DISC=0
[CRC] p=0.0  4000/10000  OK=4000 CORR=0 DISC=0
[CRC] p=0.0  5000/10000  OK=5000 CORR=0 DISC=0
[CRC] p=0.0  6000/10000  OK=6000 CORR=0 DISC=0
[CRC] p=0.0  7000/10000  OK=7000 CORR=0 DISC=0
[CRC] p=0.0  8000/10000  OK=8000 CORR=0 DISC=0
[CRC] p=0.0  9000/10000  OK=9000 CORR=0 DISC=0
[CRC] p=0.0  10000/10000  OK=10000 CORR=0 DISC=0
[FIN] CSV -> results/crc_p0.00_len8.csv  Totales: OK=10000 CORR=0 DISC=0
```

```
[CRC] p=0.05  1000/10000  OK=5 CORR=0 DISC=995
[CRC] p=0.05  2000/10000  OK=16 CORR=0 DISC=1984
[CRC] p=0.05  3000/10000  OK=25 CORR=0 DISC=2975
[CRC] p=0.05  4000/10000  OK=30 CORR=0 DISC=3970
[CRC] p=0.05  5000/10000  OK=37 CORR=0 DISC=4963
[CRC] p=0.05  6000/10000  OK=44 CORR=0 DISC=5956
[CRC] p=0.05  7000/10000  OK=53 CORR=0 DISC=6947
[CRC] p=0.05  8000/10000  OK=58 CORR=0 DISC=7942
[CRC] p=0.05  9000/10000  OK=66 CORR=0 DISC=8934
[CRC] p=0.05  10000/10000  OK=72 CORR=0 DISC=9928
[FIN] CSV -> results/crc_p0.05_len8.csv  Totales: OK=72 CORR=0 DISC=9928
esteb@Desktop-EZ:~/4to/segundo/redes/Deteccion-Correccion_1/experiments$
```

```
esteb@Desktop-EZ:~/4to/segundo/redes/Deteccion-Correccion_1/experiments$ python3 run_batch.py --alg CRC --p 0.00  --n 10
000 --len 16 --csv results/crc_p0.00_len16.csv
python3 run_batch.py --alg CRC --p 0.005 --n 10000 --len 16 --csv results/crc_p0.005_len16.csv
python3 run_batch.py --alg CRC --p 0.01  --n 10000 --len 16 --csv results/crc_p0.01_len16.csv
python3 run_batch.py --alg CRC --p 0.02  --n 10000 --len 16 --csv results/crc_p0.02_len16.csv
python3 run_batch.py --alg CRC --p 0.05  --n 10000 --len 16 --csv results/crc_p0.05_len16.csv
[CRC] p=0.0  1000/10000  OK=1000 CORR=0 DISC=0
[CRC] p=0.0  2000/10000  OK=2000 CORR=0 DISC=0
[CRC] p=0.0  3000/10000  OK=3000 CORR=0 DISC=0
[CRC] p=0.0  4000/10000  OK=4000 CORR=0 DISC=0
[CRC] p=0.0  5000/10000  OK=5000 CORR=0 DISC=0
[CRC] p=0.0  6000/10000  OK=6000 CORR=0 DISC=0
[CRC] p=0.0  7000/10000  OK=7000 CORR=0 DISC=0
[CRC] p=0.0  8000/10000  OK=8000 CORR=0 DISC=0
[CRC] p=0.0  9000/10000  OK=9000 CORR=0 DISC=0
[CRC] p=0.0  10000/10000  OK=10000 CORR=0 DISC=0
[FIN] CSV -> results/crc_p0.00_len16.csv  Totales: OK=10000 CORR=0 DISC=0
```

```
[CRC] p=0.05  1000/10000  OK=0 CORR=0 DISC=1000
[CRC] p=0.05  2000/10000  OK=1 CORR=0 DISC=1999
[CRC] p=0.05  3000/10000  OK=1 CORR=0 DISC=2999
[CRC] p=0.05  4000/10000  OK=1 CORR=0 DISC=3999
[CRC] p=0.05  5000/10000  OK=2 CORR=0 DISC=4998
[CRC] p=0.05  6000/10000  OK=2 CORR=0 DISC=5998
[CRC] p=0.05  7000/10000  OK=2 CORR=0 DISC=6998
[CRC] p=0.05  8000/10000  OK=2 CORR=0 DISC=7998
[CRC] p=0.05  9000/10000  OK=4 CORR=0 DISC=8996
[CRC] p=0.05  10000/10000  OK=4 CORR=0 DISC=9996
[FIN] CSV -> results/crc_p0.05_len16.csv  Totales: OK=4 CORR=0 DISC=9996
esteb@Desktop-EZ:~/4to/segundo/redes/Deteccion-Correccion_1/experiments$
```

```
esteb@Desktop-EZ:~/4to/segundo/redes/Deteccion-Correccion_1/experiments$ python3 run_batch.py --alg CRC --p 0.00  --n 10
000 --len 32 --csv results/crc_p0.00_len32.csv
python3 run_batch.py --alg CRC --p 0.005 --n 10000 --len 32 --csv results/crc_p0.005_len32.csv
python3 run_batch.py --alg CRC --p 0.01  --n 10000 --len 32 --csv results/crc_p0.01_len32.csv
python3 run_batch.py --alg CRC --p 0.02  --n 10000 --len 32 --csv results/crc_p0.02_len32.csv
python3 run_batch.py --alg CRC --p 0.05  --n 10000 --len 32 --csv results/crc_p0.05_len32.csv
[CRC] p=0.0  1000/10000  OK=1000 CORR=0 DISC=0
[CRC] p=0.0  2000/10000  OK=2000 CORR=0 DISC=0
[CRC] p=0.0  3000/10000  OK=3000 CORR=0 DISC=0
[CRC] p=0.0  4000/10000  OK=4000 CORR=0 DISC=0
[CRC] p=0.0  5000/10000  OK=5000 CORR=0 DISC=0
[CRC] p=0.0  6000/10000  OK=6000 CORR=0 DISC=0
[CRC] p=0.0  7000/10000  OK=7000 CORR=0 DISC=0
[CRC] p=0.0  8000/10000  OK=8000 CORR=0 DISC=0
[CRC] p=0.0  9000/10000  OK=9000 CORR=0 DISC=0
[CRC] p=0.0  10000/10000  OK=10000 CORR=0 DISC=0
[FIN] CSV -> results/crc_p0.00_len32.csv  Totales: OK=10000 CORR=0 DISC=0
```

```
[CRC] p=0.05  1000/10000  OK=0 CORR=0 DISC=1000
[CRC] p=0.05  2000/10000  OK=0 CORR=0 DISC=2000
[CRC] p=0.05  3000/10000  OK=0 CORR=0 DISC=3000
[CRC] p=0.05  4000/10000  OK=0 CORR=0 DISC=4000
[CRC] p=0.05  5000/10000  OK=0 CORR=0 DISC=5000
[CRC] p=0.05  6000/10000  OK=0 CORR=0 DISC=6000
[CRC] p=0.05  7000/10000  OK=0 CORR=0 DISC=7000
[CRC] p=0.05  8000/10000  OK=0 CORR=0 DISC=8000
[CRC] p=0.05  9000/10000  OK=0 CORR=0 DISC=9000
[CRC] p=0.05  10000/10000  OK=0 CORR=0 DISC=10000
[FIN] CSV -> results/crc_p0.05_len32.csv  Totales: OK=0 CORR=0 DISC=10000
esteb@Desktop-EZ:~/4to/segundo/redes/Deteccion-Correccion_1/experiments$
```

3. Con analyze_results.py se procesan todos los CSV y se produce una tabla global results_summary.csv con todos los datos recolectados y por medio de esta se generan gráficas de entrega correcta (%) vs *p* para cada longitud.

```
(.venv) esteb@Desktop-EZ:~/4to/segundo/redes/Deteccion-Correccion_1/experiments$ python3 analyze_results.py
    alg      p  len      N     OK  CORR   DISC  EntregaCorrecta(%)  Descartes(%)              archivo
0   CRC  0.000    8  10000  10000     0      0              100.00          0.00    crc_p0.00_len8.csv
1   CRC  0.005    4  10000   7297     0   2703               72.97         27.03   crc_p0.005_len4.csv
2   CRC  0.050   32  10000      0     0  10000                0.00        100.00   crc_p0.05_len32.csv
3   CRC  0.005   32  10000   2254     0   7746               22.54         77.46  crc_p0.005_len32.csv
4   CRC  0.050    8  10000     72     0   9928                0.72         99.28    crc_p0.05_len8.csv
5   HAM  0.000   16  10000  10000     0      0              100.00          0.00   ham_p0.00_len16.csv
6   HAM  0.005   16  10000   5036  4757    207               97.93          2.07  ham_p0.005_len16.csv
7   HAM  0.000    4  10000  10000     0      0              100.00          0.00    ham_p0.00_len4.csv
8   HAM  0.005   32  10000   2585  7084    331               96.69          3.31  ham_p0.005_len32.csv
9   CRC  0.010   32  10000    510     0   9490                5.10         94.90   crc_p0.01_len32.csv
10  HAM  0.005    8  10000   7040  2865     95               99.05          0.95   ham_p0.005_len8.csv
11  CRC  0.000   16  10000  10000     0      0              100.00          0.00   crc_p0.00_len16.csv
12  HAM  0.050    8  10000    330  7283   2387               76.13         23.87    ham_p0.05_len8.csv
13  HAM  0.020    8  10000   2308  6747    945               90.55          9.45    ham_p0.02_len8.csv
14  HAM  0.020   32  10000     69  8345   1586               84.14         15.86   ham_p0.02_len32.csv
15  HAM  0.010   16  10000   2569  6869    562               94.38          5.62   ham_p0.01_len16.csv
16  HAM  0.010    8  10000   4885  4782    333               96.67          3.33    ham_p0.01_len8.csv
17  CRC  0.020   16  10000    384     0   9616                3.84         96.16   crc_p0.02_len16.csv
18  HAM  0.000    8  10000  10000     0      0              100.00          0.00    ham_p0.00_len8.csv
19  CRC  0.000    4  10000  10000     0      0              100.00          0.00    crc_p0.00_len4.csv
20  HAM  0.050    4  10000   1452  6878   1670               83.30         16.70    ham_p0.05_len4.csv
21  HAM  0.005    4  10000   8248  1702     50               99.50          0.50   ham_p0.005_len4.csv
22  HAM  0.010   32  10000    722  8456    822               91.78          8.22   ham_p0.01_len32.csv
23  CRC  0.020    4  10000   2689     0   7311               26.89         73.11    crc_p0.02_len4.csv
24  CRC  0.010    8  10000   3741     0   6259               37.41         62.59    crc_p0.01_len8.csv
25  CRC  0.050    4  10000    380     0   9620                3.80         96.20    crc_p0.05_len4.csv
26  HAM  0.050   16  10000     57  7105   2838               71.62         28.38   ham_p0.05_len16.csv
27  HAM  0.010    4  10000   6745  3121    134               98.66          1.34    ham_p0.01_len4.csv
28  CRC  0.020    8  10000   1398     0   8602               13.98         86.02    crc_p0.02_len8.csv
29  CRC  0.010    4  10000   5224     0   4776               52.24         47.76    crc_p0.01_len4.csv
30  HAM  0.020   16  10000    675  7992   1333               86.67         13.33   ham_p0.02_len16.csv
31  CRC  0.005    8  10000   6199     0   3801               61.99         38.01   crc_p0.005_len8.csv
32  CRC  0.050   16  10000      4     0   9996                0.04         99.96   crc_p0.05_len16.csv
33  CRC  0.020   32  10000     19     0   9981                0.19         99.81   crc_p0.02_len32.csv
34  CRC  0.010   16  10000   1989     0   8011               19.89         80.11   crc_p0.01_len16.csv
35  HAM  0.020    4  10000   4614  4934    452               95.48          4.52    ham_p0.02_len4.csv
36  CRC  0.005   16  10000   4489     0   5511               44.89         55.11  crc_p0.005_len16.csv
37  HAM  0.050   32  10000     20  6841   3139               68.61         31.39   ham_p0.05_len32.csv
38  HAM  0.000   32  10000  10000     0      0              100.00          0.00   ham_p0.00_len32.csv
39  CRC  0.000   32  10000  10000     0      0              100.00          0.00   crc_p0.00_len32.csv
(.venv) esteb@Desktop-EZ:~/4to/segundo/redes/Deteccion-Correccion_1/experiments$
```

Métricas recolectadas

- status: OK, CORRECTED o DISCARD.
- Tasa de entrega correcta: (OK+CORR)/N.
- Tasa de descartes: DISC/N.

Parámetros de código

- Hamming: se calcularon bits de paridad r con $2^r \geq m + r + 1$, obteniendo tasas de código $R \approx 0.84 - 0.97$.
- CRC32: overhead fijo de 32 bits (11-50% según longitud)

Controles

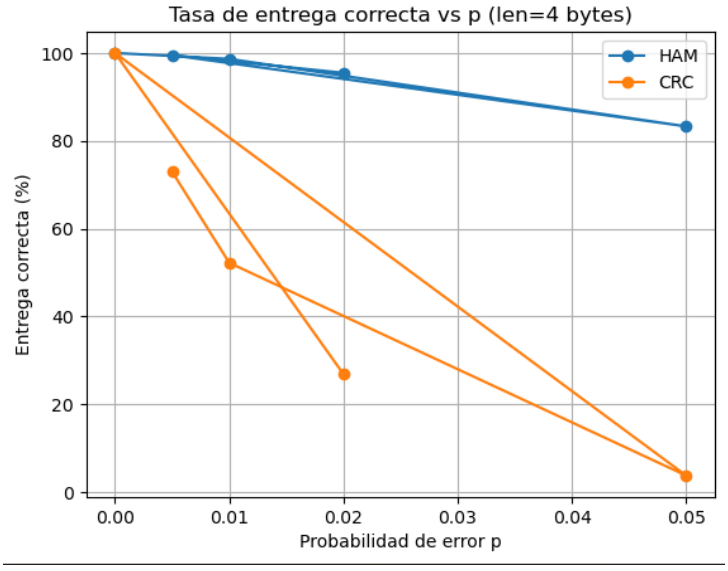
- Con $p = 0.00$ ambos alcanzan 100% de entrega.
- El comportamiento observado de CRC sigue $(1 - p)^{m+32}$.
- En Hamming, la tasa de correcciones coincide con la probabilidad de un solo error corregible.

Este procedimiento permitió obtener resultados reproducibles y comparables, que se reflejan en las tablas y gráficas.

Resultados del programa

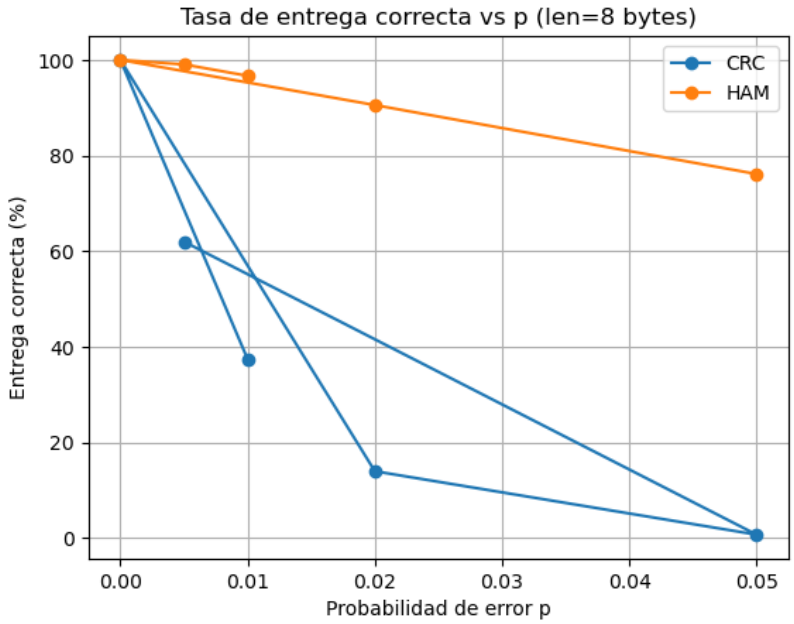
En esta sección se presentan los resultados obtenidos de las simulaciones, mostrando la tasa de entrega correcta (%) frente a la probabilidad de error del canal (p) para diferentes longitudes de mensaje. Cada gráfica compara el desempeño de los algoritmos Hamming (HAM) y CRC-32 (CRC).

Gráfica 1 – Mensajes de 4 bytes



Aquí se observa que Hamming mantiene un desempeño alto incluso con probabilidades de error de hasta 0.02, mientras que CRC decae rápidamente debido a su incapacidad de corregir errores (solo detecta).

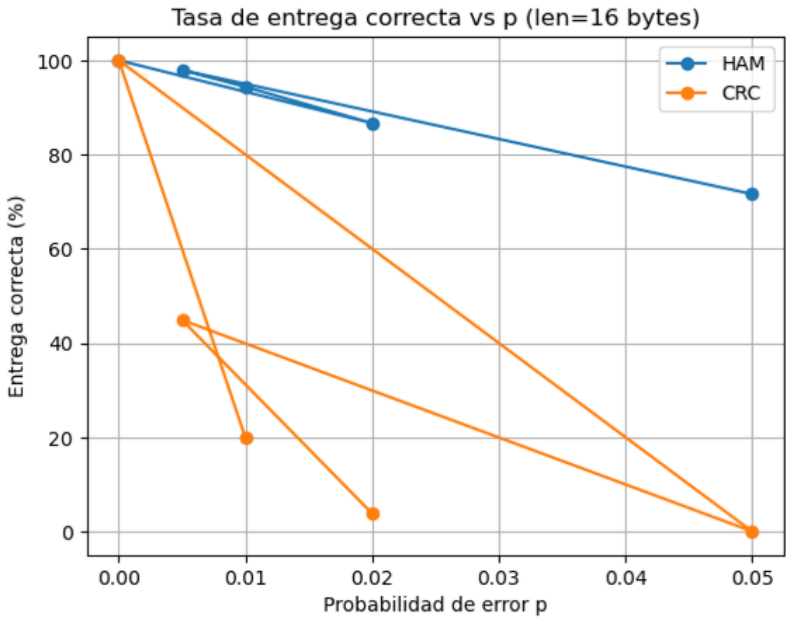
Gráfica 2 – Mensajes de 8 bytes



Hamming muestra una degradación más suave en comparación con CRC, que se reduce a tasas de entrega muy bajas cuando $p \geq 0.01$.

El overhead en Hamming es moderado, por lo que su desempeño sigue siendo superior a CRC en este rango.

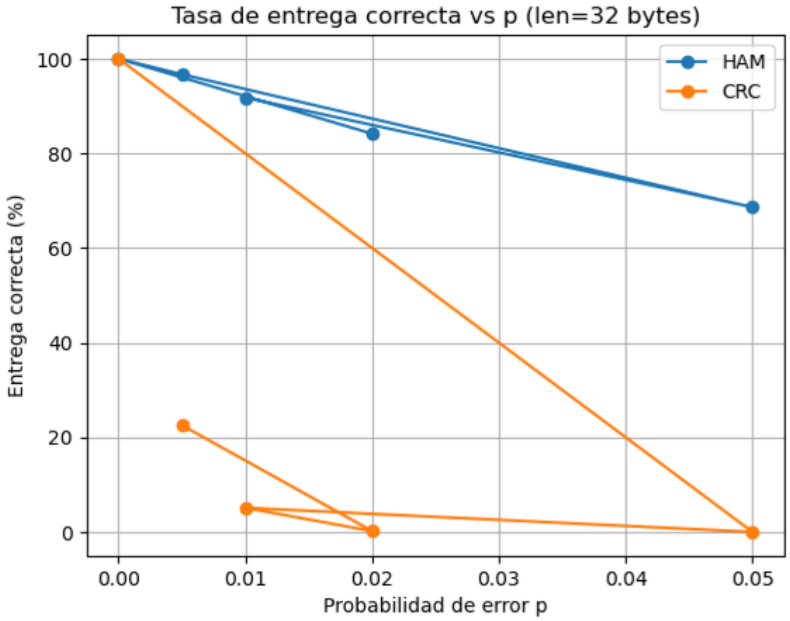
Gráfica 3 – Mensajes de 16 bytes



HAM conserva una tasa de entrega superior al 80% hasta p = 0.02, mientras que CRC cae por debajo del 20%.

Se confirma que el hecho de corregir (HAM) da ventajas claras frente a solo detectar (CRC).

Gráfica 4 – Mensajes de 32 bytes



Hamming mantiene cierta robustez, aunque la curva baja más pronunciada por la mayor longitud (más bits → más probabilidad de error).

CRC prácticamente no entrega mensajes correctos a partir de p=0.01 en adelante.

Observaciones generales

- En todos los casos, con $p = 0.00$ ambos alcanzan 100% de entrega (verificación de correcto funcionamiento).
- CRC-32 funciona bien solo en condiciones ideales o con ruido muy bajo, pero descarta gran parte de los mensajes tan pronto aumenta la probabilidad de error.
- Hamming SEC logra entregar un porcentaje mucho mayor gracias a la corrección de un solo error, aunque muestra vulnerabilidad con mensajes más largos y probabilidades de error altas ($p \geq 0.05$).
- Los resultados coinciden con la teoría: la tasa de entrega en CRC sigue $(1-p)^{m+32}$, mientras que Hamming sigue la probabilidad de que ocurra un solo error corregible.

Discusión

En las pruebas realizadas, el algoritmo de Hamming mostró un funcionamiento superior frente a CRC-32. Esto se debió a que Hamming detecta los errores, que también es capaz de corregir en ocasiones cuando ocurre una única alteración en el mensaje. En cambio, CRC únicamente detecta inconsistencias y descarta los mensajes afectados, lo cual reduce drásticamente la tasa de entrega correcta cuando aumenta la probabilidad de error. Esta diferencia quedó demostrada en las gráficas obtenidas, donde Hamming logra mantener porcentajes de entrega aceptables hasta probabilidades de error de 0.02 y CRC a diferencia colapsa mucho antes.

En cuanto a la flexibilidad para tolerar mayores tasas de error, Hamming vuelve a destacar. Este algoritmo logra recuperar mensajes incluso cuando se da el ruido moderado, esto gracias a su mecanismo de corrección de un solo bit. Aunque CRC puede detectar múltiples errores y es muy confiable en ese aspecto, su incapacidad de corregir limita su desempeño en canales con ruido significativo. Esto convierte a Hamming en una opción más robusta en entornos donde la tasa de error no es despreciable.

Por otro lado, hay escenarios donde resulta preferible utilizar un algoritmo de detección de errores en lugar de uno de corrección. Cuando el canal es altamente confiable y la probabilidad de error es muy baja, CRC puede ser más conveniente debido a su menor complejidad de implementación y debido a que garantiza una mejor detección. Además, en aplicaciones críticas donde no se acepta la mínima posibilidad de entregar información alterada, un esquema de detección como CRC ya que permite descartar el mensaje erróneo y solicitar su reenvío, lo cual asegura la integridad de los datos. En conlusión, la elección entre detección y corrección depende entre la confiabilidad del canal, los costos de retransmisión y la disponibilidad inmediata que se requiera de los datos.

Conclusiones

1. En la fase de implementación se logró desarrollar emisores y receptores funcionales para los algoritmos de Hamming (corrección) y CRC-32 (detección), cumpliendo con los requisitos del laboratorio y verificando su correcto funcionamiento con ejemplos prácticos.
2. En la fase experimental, las simulaciones mostraron que Hamming supera a CRC en escenarios con ruido moderado gracias a su capacidad de corrección, mientras que CRC resulta más adecuado en canales confiables, donde la detección basta y la simplicidad es una ventaja.
3. Las gráficas obtenidas confirmaron que la efectividad de CRC decrece rápidamente con el aumento de errores, mientras que Hamming mantiene un mayor nivel de entrega hasta que la probabilidad de error supera lo que su esquema de corrección puede manejar.

Referencias

GeeksforGeeks. (2025, May 14). *Hamming code in computer network*. GeeksforGeeks. https://www.geeksforgeeks.org/computer-networks/hamming-code-in-computer-network/

Tanenbaum, A. S., Feamster, N., & Wetherall, D. (2021). *Computer networks* (6th ed., Global edition). Pearson. https://unidel.edu.ng/focelibrary/books/103computer%20networks.pdf

Repositorio de github

https://github.com/EdwinOrtegaK/Deteccion-Correccion