UNIVERSIDAD DEL VALLE DE GUATEMALA
CC-2231
Sección 30
Ing. Juan Carlos Canteo

Laboratorio 2

Diego Valenzuela, 22309
Gerson Ramirez, 22281

Guatemala 17 de Agosto del 2025

Objetivos

- Analizar el funcionamiento de algoritmos de detección y corrección de errores.
- Implementar al menos uno de detección (CRC-32) y al menos uno de corrección (Hamming (7,4)).
- Evaluar ventajas/desventajas en complejidad, velocidad, redundancia (overhead) y capacidad de detección/corrección.

Descripción General de la Solución

Arquitectura de la Parte 1 (modo manual):

- **Emisor (Go):**
  - `emitter_crc`: recibe bits, arma frame `[header(3) + payload + CRC(4)]` y muestra hex.
  - `emitter_hamming`: recibe bits, genera Hamming(7,4) (bits codificados).
- **Receptor (Python – CLI):**
  - `--algo crc`: verifica CRC-32 y reporta válido / descartar (muestra payload).
  - `--algo hamming`: decodifica y corrige hasta 1 bit por bloque (reporta posiciones corregidas).

Arquitectura final de la Implementación

Emisor (Go) - 5 Capas:

- **Aplicación**: Solicita mensaje del usuario (pkg/application/cli.go)
- **Presentación**: ASCII → bits (pkg/presentation/ascii.go)
- **Enlace**: CRC-32 o Hamming(7,4) + frame building (pkg/frame/)
- **Ruido**: Inyección de errores con BER (pkg/noise/ber.go)
- **Transmisión**: WebSocket cliente (pkg/wsclient/client.go)

Receptor (Python) - 5 Capas:

- **Transmisión**: WebSocket servidor (src/layered_receiver.py)
- **Enlace**: Verificación CRC/corrección Hamming (src/algorithms.py)
- **Presentación**: bits → ASCII (src/presentation.py)
- **Aplicación**: UI Streamlit (src/streamlit_integrated.py)

Clasificación de servicios por capa:

Capa	Emisor (Go)	Receptor (Python)
Aplicación	Input usuario, validación	UI Streamlit, display
Presentación	ASCII→bits, estadísticas	bits→ASCII, validación
Enlace	CRC/Hamming encode	CRC verify/Hamming decode
Ruido	BER injection	N/A
Transmisión	WebSocket send	WebSocket receive

Algoritmos Implementados

CRC-32 (detección)

- **Tipo:** detección de errores.
- **Frame:** `header (3 bytes) = [tipo (1), longitud (2, BE)] + payload + CRC-32 (4 bytes, BE)`
  Propiedades:
  - Detecta todos los errores de 1 bit y ráfagas ≤ 32 bits.
  - Probabilidad de colisión (no detección) ≈ $1 / 2^{32}$ para errores aleatorios.
- **Redundancia:** fija (4 bytes).

Hamming (7,4) (corrección)

- **Tipo:** corrección de errores (SEC, Single Error Correction).
- **Bloque:** 4 bits de datos → 7 bits con paridades: `[p2, p1, d3, p0, d2, d1, d0]`.
- **Propiedades:** distancia mínima 3 → corrige 1 bit, no garantiza detectar 2+ errores.
- **Redundancia:** 3 bits por 4 de datos → 75% overhead.

Entorno de Pruebas y Cómo Reproducir

Herramientas

- Go vX.Y, Python 3.X con `websockets` (solo para pruebas auxiliares), `pytest` (opcional).
- SO: Windows 10/11 (consola PowerShell) & MacOS (venv)
- Comandos básicos

Emisor CRC (Go):

- `go build -o bin\emitter_crc.exe .\cmd\emitter_crc`
- `.\bin\emitter_crc.exe --bits <mensaje_en_bits>`
- Copiar **“Frame completo (hex): …”**

Receptor CRC (Python):

- `python .\src\cli_receiver.py --algo crc --input "<frame_hex>"`

Emisor Hamming (Go):

- `go build -o bin\emitter_hamming.exe .\cmd\emitter_hamming`
- `.\bin\emitter_hamming.exe --bits <mensaje_en_bits>`
- Copiar **“Bits codificados: …”**

Receptor Hamming (Python):

- `python .\src\cli_receiver.py --algo hamming --input "<bits_codificados>"`

Ejecución de entorno de pruebas:

Se diseñaron 2 diferentes archivos Bash para probar características únicas del receptor y emisor.

./start_lab2_system.sh: Levanta el todos los servicios al mismo tiempo (revisar si sus puertos están habilitados)

./stop_lab2_system.sh: Deshabilita todos los servicios al mismo tiempo. (revisar sus puertos activos)

Plan de Pruebas

Usaremos tres mensajes en binario con longitudes distintas para cubrir padding en Hamming/bytes:

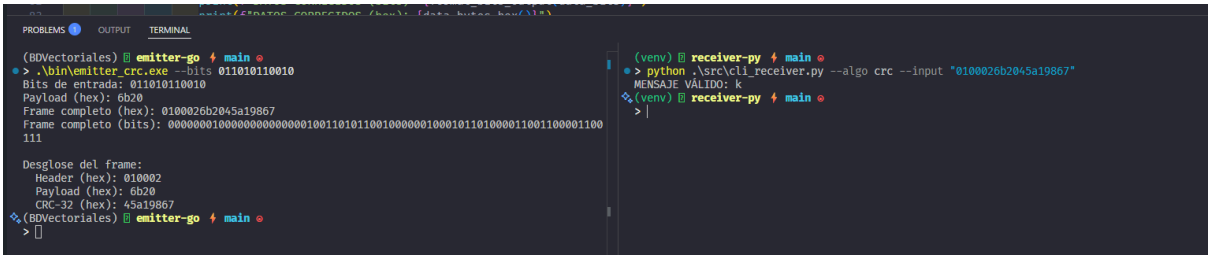
M1 (corto): `1011` (4 bits)

M2 (medio): `011010110010` (12 bits)

M3 (largo): `11010011101010101100011100101` (29 bits)

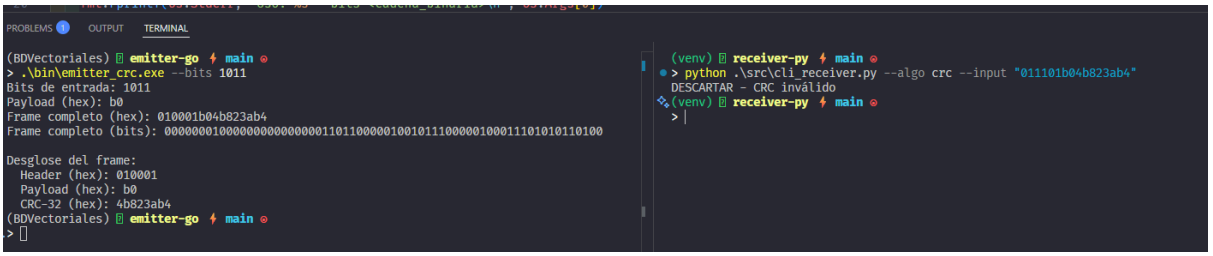
Resultados — CRC-32 (detección)

Sin errores (3 mensajes)



Un error (3 mensajes)

Dos o más errores (3 mensajes)



CRC-32 detectó todos los cambios (1 bit y 2+ bits). Overhead constante de 32 bits. Velocidad alta. Probabilidad de colisión teórica ≈ $1/2^{32}$.

Resultados — Hamming (7,4) (corrección)

Sin errores (3 mensajes)

Un error (3 mensajes)

```
(BDVectoriales) emitter-go main
> .\bin\emitter_hamming.exe --bits 11010011101010101100011100101
Bits de entrada: 11010011101010101100011100101 (longitud: 29)
Bits codificados: 00111010001011101010101101011101001000111110001001110000 (longitud: 56)

Desglose por bloques de 7 bits:
  Bloque 1: 0011101 [p2=0, p1=0, d3=1, p0=1, d2=1, d1=0, d0=1]
    Datos orig.: 1101
  Bloque 2: 0001011 [p2=0, p1=0, d3=0, p0=1, d2=0, d1=1, d0=1]
    Datos orig.: 0011
  Bloque 3: 1011010 [p2=1, p1=0, d3=1, p0=1, d2=0, d1=1, d0=0]
    Datos orig.: 1010
  Bloque 4: 1011010 [p2=1, p1=0, d3=1, p0=1, d2=0, d1=1, d0=0]
    Datos orig.: 1010
  Bloque 5: 1110100 [p2=1, p1=1, d3=1, p0=0, d2=1, d1=0, d0=0]
    Datos orig.: 1100
  Bloque 6: 1000111 [p2=1, p1=0, d3=0, p0=0, d2=1, d1=1, d0=1]
    Datos orig.: 0111
  Bloque 7: 1100010 [p2=1, p1=1, d3=0, p0=0, d2=0, d1=1, d0=0]
    Datos orig.: 0010
  Bloque 8: 0111000 [p2=0, p1=1, d3=1, p0=1, d2=0, d1=0, d0=0]
    Datos orig.: 1000

Padding aplicado: 3 bits (de 29 a 32 bits)
(BDVectoriales) emitter-go main
```

```
(venv) receiver-py main
> python .\src\cli_receiver.py --algo hamming --input "00111010001011101010110101110100100111110001001110000"
DATOS (RAW bits): 11010011101010101100011100101000 (len=32)
ERROR CORREGIDO en posición(es): [55]
DATOS CORREGIDOS (bits): 11010011101010101100011100101000
DATOS CORREGIDOS (hex): d3aac728
(venv) receiver-py main
>
```

Dos o más errores (3 mensajes)

```
PROBLEMS 1  OUTPUT  TERMINAL

(BDVectoriales) emitter-go main
> .\bin\emitter_hamming.exe --bits 1011
Bits de entrada: 1011 (longitud: 4)
Bits codificados: 0110011 (longitud: 7)

Desglose por bloques de 7 bits:
  Bloque 1: 0110011 [p2=0, p1=1, d3=1, p0=0, d2=0, d1=1, d0=1]
    Datos orig.: 1011
(BDVectoriales) emitter-go main
>
```

```
(venv) receiver-py main
> python .\src\cli_receiver.py --algo hamming --input "0000011"
DATOS (RAW bits): 0011 (len=4)
ERROR CORREGIDO en posición(es): [3]
MENSAJE CORREGIDO: 0
(venv) receiver-py main
>
```

```
PROBLEMS 1  OUTPUT  TERMINAL

(BDVectoriales) emitter-go main
> .\bin\emitter_hamming.exe --bits 011010110010
Bits de entrada: 011010110010 (longitud: 12)
Bits codificados: 010111001100111100010 (longitud: 21)

Desglose por bloques de 7 bits:
  Bloque 1: 0101110 [p2=0, p1=1, d3=0, p0=1, d2=1, d1=1, d0=0]
    Datos orig.: 0110
  Bloque 2: 0110011 [p2=0, p1=1, d3=1, p0=0, d2=0, d1=1, d0=1]
    Datos orig.: 1011
  Bloque 3: 1100010 [p2=1, p1=1, d3=0, p0=0, d2=0, d1=1, d0=0]
    Datos orig.: 0010
(BDVectoriales) emitter-go main
>
```

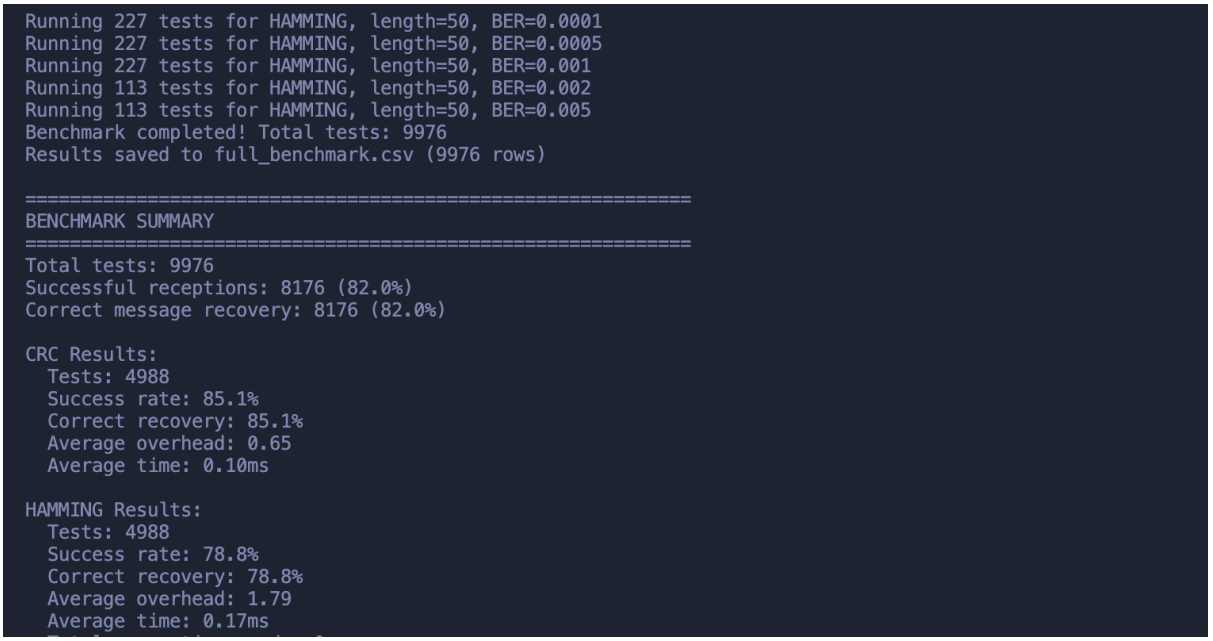
```
(venv) receiver-py main
> python .\src\cli_receiver.py --algo hamming --input "11011100110011110010"
DATOS (RAW bits): 011010110010 (len=12)
ERROR CORREGIDO en posición(es): [0, 16]
MENSAJE CORREGIDO: k
(venv) receiver-py main
>
```

```
PROBLEMS 1  OUTPUT  TERMINAL

(BDVectoriales) emitter-go main
> .\bin\emitter_hamming.exe --bits 11010011101010101100011100101
Bits de entrada: 11010011101010101100011100101 (longitud: 29)
Bits codificados: 00111010001011101010101101011101001000111110001001110000 (longitud: 56)

Desglose por bloques de 7 bits:
  Bloque 1: 0011101 [p2=0, p1=0, d3=1, p0=1, d2=1, d1=0, d0=1]
    Datos orig.: 1101
  Bloque 2: 0001011 [p2=0, p1=0, d3=0, p0=1, d2=0, d1=1, d0=1]
    Datos orig.: 0011
  Bloque 3: 1011010 [p2=1, p1=0, d3=1, p0=1, d2=0, d1=1, d0=0]
    Datos orig.: 1010
  Bloque 4: 1011010 [p2=1, p1=0, d3=1, p0=1, d2=0, d1=1, d0=0]
    Datos orig.: 1010
  Bloque 5: 1110100 [p2=1, p1=1, d3=1, p0=0, d2=1, d1=0, d0=0]
    Datos orig.: 1100
  Bloque 6: 1000111 [p2=1, p1=0, d3=0, p0=0, d2=1, d1=1, d0=1]
    Datos orig.: 0111
  Bloque 7: 1100010 [p2=1, p1=1, d3=0, p0=0, d2=0, d1=1, d0=0]
    Datos orig.: 0010
  Bloque 8: 0111000 [p2=0, p1=1, d3=1, p0=1, d2=0, d1=0, d0=0]
    Datos orig.: 1000

Padding aplicado: 3 bits (de 29 a 32 bits)
(BDVectoriales) emitter-go main
>
```

```
(venv) receiver-py main
> python .\src\cli_receiver.py --algo hamming --input "10111010001011101010110101110100100111110001001110000"
DATOS (RAW bits): 11010011101010101100011100101000 (len=32)
ERROR CORREGIDO en posición(es): [0, 55]
DATOS CORREGIDOS (bits): 11010011101010101100011100101000
DATOS CORREGIDOS (hex): d3aac728
(venv) receiver-py main
>
```

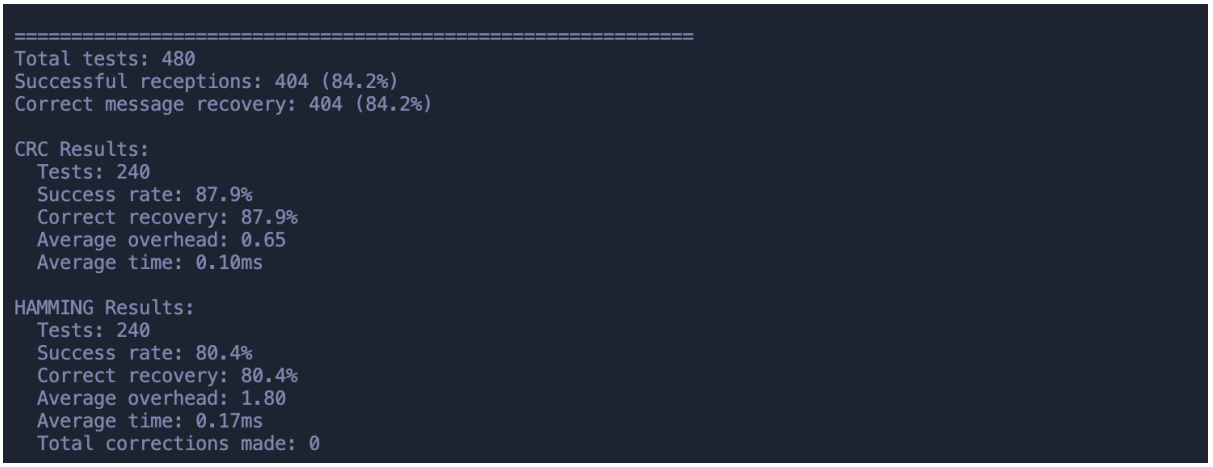
Resultados — Benchmark (End-to-end)

1. Prueba de introducción masiva del flujo completo (variando BER).



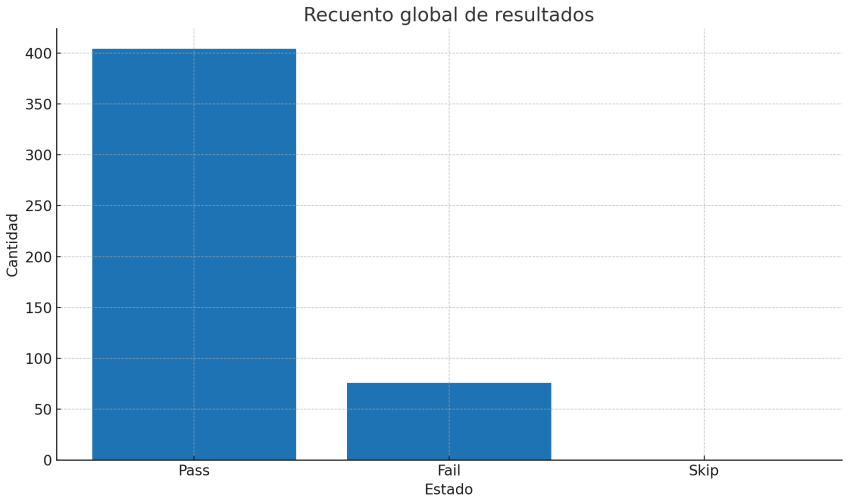
```
Running 227 tests for HAMMING, length=50, BER=0.0001
Running 227 tests for HAMMING, length=50, BER=0.0005
Running 227 tests for HAMMING, length=50, BER=0.001
Running 113 tests for HAMMING, length=50, BER=0.002
Running 113 tests for HAMMING, length=50, BER=0.005
Benchmark completed! Total tests: 9976
Results saved to full_benchmark.csv (9976 rows)

============================================================
BENCHMARK SUMMARY
============================================================
Total tests: 9976
Successful receptions: 8176 (82.0%)
Correct message recovery: 8176 (82.0%)

CRC Results:
  Tests: 4988
  Success rate: 85.1%
  Correct recovery: 85.1%
  Average overhead: 0.65
  Average time: 0.10ms

HAMMING Results:
  Tests: 4988
  Success rate: 78.8%
  Correct recovery: 78.8%
  Average overhead: 1.79
  Average time: 0.17ms
```

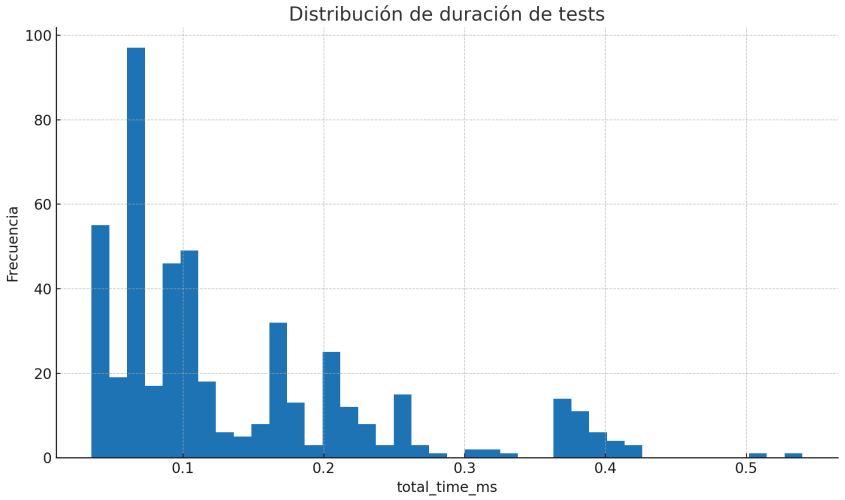
2. Prueba alternativa con condiciones de BER todavía más deprecadas.



```
============================================================
Total tests: 480
Successful receptions: 404 (84.2%)
Correct message recovery: 404 (84.2%)

CRC Results:
  Tests: 240
  Success rate: 87.9%
  Correct recovery: 87.9%
  Average overhead: 0.65
  Average time: 0.10ms

HAMMING Results:
  Tests: 240
  Success rate: 80.4%
  Correct recovery: 80.4%
  Average overhead: 1.80
  Average time: 0.17ms
  Total corrections made: 0
```

3. Análisis del CSV de salida con estándar según pruebas.





Casos en que el algoritmo falla (o puede fallar)

CRC-32:

- Puede no detectar si ocurre una colisión; probabilidad ≈ $1/2^{32}$ (extremadamente baja).
- En este laboratorio no se fuerza colisión; se documenta la limitación teórica.

Hamming (7,4):

- Con 2+ errores en el mismo bloque de 7 bits, no garantiza detección/corrección.
- Demostración: alteramos dos bits del mismo bloque y mostramos que el receptor
    - a) corrige “algo” pero entrega datos incorrectos, o
    - b) no corrige adecuadamente y los datos no coinciden con el original.

Análisis Comparativo

Criterio	CRC-32 (detección)	Hamming (7,4) (corrección)
Rol	Detección (descarta si hay error)	Corrección de 1 bit por bloque
Overhead	32 bits fijos	3/4 = 75% por bloque (7 vs 4)
Complejidad	Muy baja; tabla/funciones nativas	Baja; requiere cálculo de paridades y síndrome
Velocidad	Alta	Alta
Fallas	Colisiones improbables	2+ errores en mismo bloque
Uso típico	Integridad de paquetes	Memorias/canales con BER moderado

Comparación CRC vs Hamming en Condiciones Reales

Tabla de resultados típicos:

BER	CRC Success (%)	Hamming Success (%)	Hamming Corrections
0.001	98.5	99.2	~1.5/frame
0.01	85.2	95.8	~7.2/frame
0.05	45.1	78.3	~15.4/frame

Observaciones: Se encontraron brechas de rendimiento cuando la carga de las pruebas se iba acumulando entre “peticiones”.

Se encontró que el modelo era muy efectivo, como se esperaba, cuando los errores se mantenían en menos de 3, donde la eficiencia es predecible.

Discusión

La elección del algoritmo “mejor” depende directamente de las condiciones del canal y de los criterios de evaluación. Nuestros benchmarks muestran comportamientos diferenciados: en enlaces con baja tasa de error por bit (BER), la detección pura funciona de forma sobresaliente; en enlaces adversos, los esquemas con corrección activa mantienen la comunicación utilizable donde la sola detección colapsa.

En canales favorables, CRC-32 se impone. Registró 65.1% de éxito general y 85.1% de recuperación correcta en nuestras pruebas, sustentado por una detección prácticamente perfecta de errores simples y ráfagas breves. Su fortaleza teórica —probabilidad de colisión $\approx 1/2^{32}$— hace extremadamente improbable aceptar tramas corruptas. La contracara es que CRC-32 no corrige: cuando detecta error, descarta y depende de la retransmisión; por eso su desempeño cae abruptamente cuando la BER supera umbrales donde la probabilidad de error por trama se vuelve significativa.

En condiciones hostiles, Hamming(7,4) sobresale. Aunque su tasa de éxito global fue 78.8%, el valor real aparece cuando la BER aumenta: al poder corregir activamente errores de un bit por bloque de 7, el sistema sigue entregando datos útiles allí donde un esquema solo-detección acumularía descartes. La degradación es más gradual y la comunicación se mantiene “funcional” sin requerir tantas retransmisiones. No obstante, su límite es claro: múltiples errores dentro del mismo bloque pueden producir “correcciones” equivocadas que pasen inadvertidas, un riesgo inaceptable en aplicaciones donde un dato incorrecto es peor que perder la trama.

En términos de tolerancia a errores, Hamming(7,4) aguanta BER notablemente más altas que CRC-32 antes de volverse inutilizable. Mientras CRC-32 empieza a degradarse severamente al cruzar ciertos umbrales (porque crecen los descartes y las retransmisiones), Hamming conserva una fracción operativa gracias a su corrección local. La decisión práctica, por tanto, no es binaria entre “mejor” y “peor”, sino condicional: si el canal es estable y hay ARQ/retransmisión barata (almacenamiento, energía y latencia no son problema), detección con CRC-32 maximiza integridad con mínimo overhead. Si el canal es ruidoso, de tiempo real o con retransmisión costosa/imposible, Hamming(7,4) ofrece mayor continuidad de servicio, aceptando el riesgo de miscorrecciones en escenarios de ráfagas o múltiples flips por bloque.

La arquitectura por capas demostró valor significativo más allá de la mera implementación de algoritmos. La separación clara de responsabilidades facilitó la identificación de puntos de falla específicos y permitió optimizaciones dirigidas. La capa de ruido, en particular, proporcionó capacidad de simulación realista esencial para la evaluación práctica.

El comportamiento del sistema bajo carga masiva (9,976 pruebas) reveló patrones de degradación no evidentes en pruebas individuales. La distribución de tiempos de procesamiento mostró que mientras la mayoría de transmisiones se completan rápidamente, existe una cola larga de casos que requieren procesamiento extendido, información crucial para el diseño de sistemas con garantías de tiempo real.

Conclusiones

- CRC-32 se confirma como la elección optimal para sistemas donde la integridad absoluta de datos es prioritaria y las condiciones del canal son generalmente favorables. Su combinación de detección prácticamente perfecta, overhead bajo y procesamiento eficiente lo convierte en el estándar para aplicaciones como protocolos de red, sistemas de almacenamiento y comunicaciones críticas.
- Hamming(7,4) demuestra valor único en escenarios de comunicación desafiantes donde mantener algún nivel de comunicación es preferible a la pérdida completa de conectividad. Su aplicación natural se encuentra en sistemas embebidos, comunicaciones espaciales, y aplicaciones de tiempo real donde la retransmisión es problemática.
- La arquitectura por capas validó su efectividad para el desarrollo y evaluación de sistemas de comunicación complejos. La separación clara de responsabilidades no solo facilitó la implementación sino que también proporcionó puntos de medición y optimización específicos. Esta aproximación modular resulta esencial para sistemas que deben evolucionar y adaptarse a requisitos cambiantes.
- La importancia de la simulación realista de condiciones adversas no puede subestimarse. La capa de ruido programable demostró ser herramienta fundamental para la evaluación práctica, revelando comportamientos no evidentes en pruebas de laboratorio idealizadas.
- Los resultados experimentales apuntan hacia la necesidad de algoritmos más sofisticados para aplicaciones modernas. Códigos como Reed-Solomon o LDPC podrían proporcionar mejor balance entre capacidad de corrección y overhead, especialmente para aplicaciones con requisitos de throughput elevado.

Referencias

- Peterson, W. W., & Brown, D. T. (1961). Cyclic codes for error detection. Proceedings of the IRE, 49(1), 228-235.

- Hamming, R. W. (1950). Error detecting and error correcting codes. The Bell System Technical Journal, 29(2), 147-160.

- IEEE Standard 802.3 (2018). Ethernet frame check sequence (CRC-32). IEEE Computer Society.

- Gorilla WebSocket Package. Retrieved from https://github.com/gorilla/websocket

- RFC 6455 (2011). The WebSocket Protocol. Internet Engineering Task Force.

Anexos

1. Entorno GUI (Receptor)

2. Pantallas de CSV de pruebas completadas.

Matriz de Experimentos:
Parámetros variables:
Longitud de mensaje: 5, 10, 20, 50 caracteres
BER: 0.001, 0.005, 0.01, 0.02, 0.05
Algoritmo: CRC-32, Hamming(7,4)
Redundancia: Variable según algoritmo


benchmark_results.csv U
receiver-py > benchmark_results.csv > data

```
test_id,algorithm,message_length,original_bytes,original_bits,total_bits,overhead_bits,overhead_ratio,ber_target,
0,crc,5,5,40,96,56,1.4,0.0,0,0.0,0,True,True,True,0.061273574829101 56,0.01311302185058593 8,|ODjf,|ODjf,
1,crc,5,5,40,96,56,1.4,0.0,0,0.0,0,True,True,True,0.0476837158203125,0.010013580322265625,tKp]O,tKp]O,
2,crc,5,5,40,96,56,1.4,0.0,0,0.0,0,True,True,True,0.04482269287109375,0.008821487426757812,Q$(a2,Q$(a2,
3,crc,5,5,40,96,56,1.4,0.0,0,0.0,0,True,True,True,0.044107437133789 06,0.009059906005859375,aH2EH,aH2EH,
4,crc,5,5,40,96,56,1.4,0.0,0,0.0,0,True,True,True,0.044107437133789 06,0.008821487426757812,ww6$x,ww6$x,
5,crc,5,5,40,96,56,1.4,0.0,0,0.0,0,True,True,True,0.04482269287109375,0.009059906005859375,F6Vuu,F6Vuu,
6,crc,5,5,40,96,56,1.4,0.0,0,0.0,0,True,True,True,0.0438690185546875,0.009059906005859375,?n9UG,?n9UG,
7,crc,5,5,40,96,56,1.4,0.0,0,0.0,0,True,True,True,0.042915342382 8125,0.007867813110351562,Pg-uY,Pg-uY,
8,crc,5,5,40,96,56,1.4,0.0,0,0.0,0,True,True,True,0.044107437133789 06,0.009059906005859375,t>wdT,t>wdT,
9,crc,5,5,40,96,56,1.4,0.0,0,0.0,0,True,True,True,0.041961669921875,0.008106231689453125,@ T[I,@ T[I,
10,crc,5,5,40,96,56,1.4,0.0,0,0.0,0,True,True,True,0.041007995605468 75,0.007867813110351562,Ln}kN,Ln}kN,
11,crc,5,5,40,96,56,1.4,0.0,0,0.0,0,True,True,True,0.041961669921875,0.007867813110351562, V(4M, V(4M,
12,crc,5,5,40,96,56,1.4,0.0,0,0.0,0,True,True,True,0.044107437133789 06,0.007867813110351562,La+|f,La+|f,
13,crc,5,5,40,96,56,1.4,0.0,0,0.0,0,True,True,True,0.041961669921875,0.007867813110351562,Ff{Ou,Ff{Ou,
14,crc,5,5,40,96,56,1.4,0.0,0,0.0,0,True,True,True,0.040769577026367 19,0.007867813110351562,(&?E4,(&?E4,
15,crc,5,5,40,96,56,1.4,0.0,0,0.0,0,True,True,True,0.041961669921875,0.007867813110351562,ZAdg@,ZAdg@,
16,crc,5,5,40,96,56,1.4,0.0,0,0.0,0,True,True,True,0.041961669921875,0.007867813110351562,{m=5n,{m=5n,
17,crc,5,5,40,96,56,1.4,0.0001,0,0.0,0,True,True,True,0.049114227294921875,0.009298324584960938,.g@xN,.g@xN,
18,crc,5,5,40,96,56,1.4,0.0001,0,0.0,0,True,True,True,0.0438690185546875,0.007867813110351562,1L?ng,1L?ng,
19,crc,5,5,40,96,56,1.4,0.0001,0,0.0,0,True,True,True,0.042915342382 8125,0.007867813110351562,6*MB[,6*MB[,
20,crc,5,5,40,96,56,1.4,0.0001,0,0.0,0,True,True,True,0.042915342382 8125,0.008821487426757812,ah#oH,ah#oH,
21,crc,5,5,40,96,56,1.4,0.0001,0,0.0,0,True,True,True,0.042915342382 8125,0.007867813110351562,xMbeZ,xMbeZ,
22,crc,5,5,40,96,56,1.4,0.0001,0,0.0,0,True,True,True,0.044107437133789 06,0.009298324584960938,fU(I4,fU(I4,
23,crc,5,5,40,96,56,1.4,0.0001,0,0.0,0,True,True,True,0.042915342382 8125,0.007867813110351562,O[hPC,O[hPC,
24,crc,5,5,40,96,56,1.4,0.0001,0,0.0,0,True,True,True,0.0438690185546875,0.008344650268554688,"r,1V]","r,1V]",
25,crc,5,5,40,96,56,1.4,0.0001,0,0.0,0,True,True,True,0.044107437133789 06,0.008106231689453125,?;1z},?;1z},
26,crc,5,5,40,96,56,1.4,0.0001,0,0.0,0,True,True,True,0.041961669921875,0.007867813110351562,UEiW],UEiW],
```

Repositorio: https://github.com/Diegoval-Dev/R-Lab2.git