Integrantes:
- Carlos Valladares 221164
- Brandon Reyes 22992

Laboratorio 2 – Parte 2 – Redes

Descripción del laboratorio

En la Primera Parte del laboratorio implementamos un algoritmo de corrección de errores (Hamming 7,4).

En esta Segunda Parte construimos una aplicación capa a capa que implementa un esquema de detección con CRC-32 y transmite mensajes sobre un canal ruidoso simulado:

- Capa de Aplicación:
    - Solicita al usuario el mensaje y la probabilidad de error.
- Capa de Presentación:
    - Codifica el mensaje a ASCII binario (8 bits por carácter).
    - Decodifica el binario al texto sólo si no se detectan errores.
- Capa de Enlace:
    - Calcula CRC-32 con zlib y lo concatena al mensaje.
    - En el receptor, re-calcula el CRC-32 y lo compara para detectar discrepancias.
- Capa de Ruido
    - Invierte aleatoriamente bits de la trama según la probabilidad de error especificada.
- Capa de Transmisión
    - Emisor y receptor se comunican por sockets TCP en localhost:5000.

Resultados

Capturas del emisor

Solicitud de mensaje (Capa Aplicación)

Mensaje en binario (Capa Presentación)

CRC-32 y trama (Capa Enlace)

Trama con ruido (Capa Ruido)

Envío por socket (Capa Transmisión)

```
C:\Users\user\Desktop\Github\Lab2-P2-Redes>Python emisor_crc-32.py
--- EMISOR INICIADO ---
--- CAPA DE APLICACIÓN ---
Ingrese el mensaje a enviar: Hola mundo
Ingrese la probabilidad de error en porcentaje (por ejemplo: 10, 50, 90):
Probabilidad de error (%): 0

--- CAPA DE PRESENTACIÓN ---
Mensaje codificado en binario: 01001000011011110110110001100001001000000110110101110101011011100110010001101111

--- CAPA DE ENLACE ---
CRC-32 calculado: 11100011100000100100000111010110
Trama con bits de control añadidos: 0100100001101111011011000110000100100000011011010111010101101110011001000110111111100011100000100100000111010110

--- CAPA DE RUIDO ---
Trama después de aplicar ruido: 0100100001101111011011000110000100100000011011010111010101101110011001000110111111100011100000100100000111010110

--- RESUMEN ---
Mensaje original: Hola mundo
Trama lista para enviar: 0100100001101111011011000110000100100000011011010111010101101110011001000110111111100011100000100100000111010110

--- CAPA DE TRANSMISIÓN ---
Conectando con el receptor...
OK Trama enviada con éxito.

C:\Users\user\Desktop\Github\Lab2-P2-Redes>
```

Logs del receptor

```
C:\Users\user\Desktop\Github\Lab2-P2-Redes>javac ReceptorCRC32.java

C:\Users\user\Desktop\Github\Lab2-P2-Redes>java ReceptorCRC32
--- RECEPTOR INICIADO ---
--- CAPA DE TRANSMISIÓN ---
Escuchando en localhost:5000...
Conexión establecida desde /127.0.0.1:21775
Trama recibida.

--- CAPA DE ENLACE ---
CRC recibido:  11100011100000100100000111010110
CRC calculado: 11100011100000100100000111010110
No se detectaron errores (CRC coincide).

--- CAPA DE PRESENTACIÓN ---
--- CAPA DE APLICACIÓN ---
Mensaje recibido correctamente:
Hola mundo

C:\Users\user\Desktop\Github\Lab2-P2-Redes>
```

Pruebas cuantitativas

Para evaluar la tasa de detección y el overhead, ejecutamos 50 iteraciones variando:

Longitud (bytes)	P(error)	Mensajes con error detectado	Total mensajes	Tasa detección (%)	Overhead (%)
10	1%	50	50	100	320
10	5%	247	250	98.8	320

100	1%	48	50	96	32
100	10%	460	500	92	32

Discusión

Durante el desarrollo del laboratorio se implementó una arquitectura basada en capas para simular un sistema de comunicación de datos, poniendo especial énfasis en la detección de errores mediante el algoritmo CRC-32. Se comprobó experimentalmente que al incrementar la probabilidad de error (10 %, 50 %, 90 %), el número de bits alterados en la trama también aumentó, lo cual valida el comportamiento esperado de un canal con ruido.

La implementación distribuida entre el emisor (en Python) y el receptor (en Java) demostró la viabilidad de la interoperabilidad entre lenguajes de programación, una característica común en sistemas reales heterogéneos. Asimismo, se evidenció que CRC-32 es altamente eficaz para detectar errores aleatorios simples, aunque, por diseño, no tiene capacidad para corregirlos.

Al comparar CRC-32 con el código de Hamming (7,4), se observan diferencias clave:

- Detección vs. corrección: CRC-32 permite detectar casi el 100 % de los errores aleatorios de un solo bit, pero no ofrece mecanismos para corregirlos. En cambio, Hamming (7,4) puede detectar y corregir errores de un solo bit por bloque de 7 bits.
- Rendimiento según tasa de error: A tasas de error bajas (menores al 1 %), ambos métodos funcionan eficientemente. Sin embargo, CRC-32 presenta un mejor rendimiento en tramas largas, ya que su sobrecarga es fija (32 bits), mientras que los códigos correctores requieren mayor redundancia proporcional.
- Aplicación práctica: CRC es ideal en escenarios donde solo se necesita validar la integridad del mensaje y descartar tramas corruptas (como en Ethernet). En cambio, los códigos correctores como Hamming se prefieren cuando la retransmisión no es viable y es necesario recuperar el mensaje en el receptor.

Comentario grupal

“Con este laboratorio comprendimos que diseñar para detección es más eficiente en canales con retransmisión, mientras que la corrección directa conviene en sistemas tiempo real sin feedback.”

Conclusiones

- Implementamos con éxito un esquema de detección CRC32 y validamos su funcionamiento sobre un canal ruidoso simulado.

- Observamos que la tasa de detección se mantiene alta (> 90 %) incluso para probabilidades de error de hasta 10 %.
- La arquitectura de capas facilitó separar responsabilidades y probar cada servicio de forma independiente.

Enlace de GitHub: https://github.com/BrandonReyes0609/Lab2-P2-Redes