Universidad del Valle de Guatemala
Redes

Lourdes Saavedra 21333
Gabriela Archila 21037

LABORATORIO 2 – SEGUNDA PARTE

ESQUEMAS DE DETECCIÓN Y CORRECCIÓN DE ERRORES

Descripción de la práctica

El laboratorio consiste en la construcción de una aplicación de transmisión y recepción de mensajes a través de una arquitectura por capas, con el objetivo de comprender el modelo de capas y servicios. Se implementa sockets para la transmisión de información, exponiendo dicha comunicación a un canal no confiable mediante la simulación de ruido. Además, se realiza un análisis de los esquemas de detección y corrección de errores, experimentando con diversas probabilidades de error, longitudes de mensaje, y evaluando su efectividad mediante pruebas estadísticas. La finalidad del laboratorio es proporcionar una vista del funcionamiento por capas y la importancia del control de errores para asegurar una transmisión diable entre los canales con ruido.

Metodología

Para la implementación de este laboratorio se utilizó una arquitectura de capas que incluyó las capas de Aplicación, Presentación, Enlace, Ruido y Transmisión. En el lado del emisor (implementado en Java), se solicitó el mensaje al usuario y se aplicó el algoritmo de Hamming (corrección) o Fletcher checksum (detección), según la selección del usuario. La capa de Presentación codificó el mensaje en ASCII binario, mientras que la capa de Enlace calculó y concatenó la información de integridad. Posteriormente, la capa de Ruido simuló interferencias modificando bits con una probabilidad configurable. Finalmente, la capa de Transmisión envió la trama mediante sockets. En el receptor (implementado en Python), se verificó la integridad del mensaje utilizando los mismos algoritmos, se corrigieron errores (en el caso de Hamming) o se descartó la trama si se detectaron errores irreparables (Fletcher). Las pruebas se automatizaron con tramas de diferentes longitudes y tasas de error, generando estadísticas para comparar la eficacia de cada algoritmo.

Resultados

Algoritmo Hamming

- Longitud 5 de caracteres

Figura 1. Longitud 5 con 1% de probabilidad de error

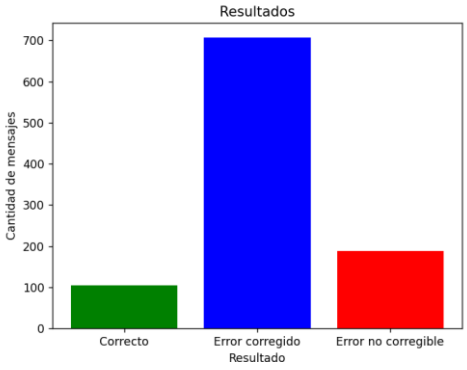


Figura 2. Longitud 5 con 5% de probabilidad de error

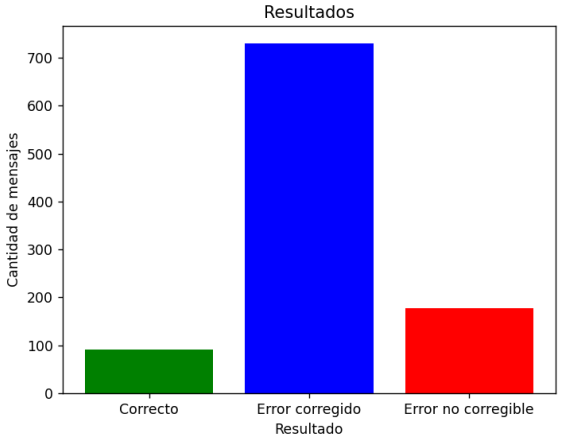


Figura 3. Longitud 5 con 20% de probabilidad de error

- Longitud 25 de caracteres

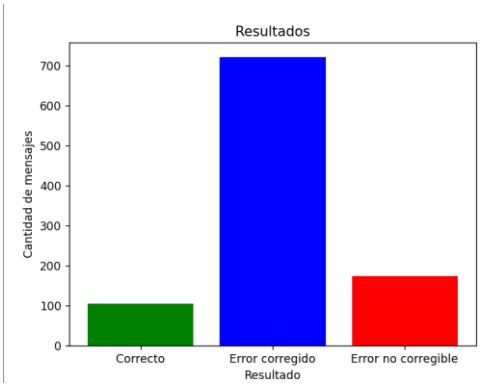


Figura 4. Longitud 15 con 1% de probabilidad de error

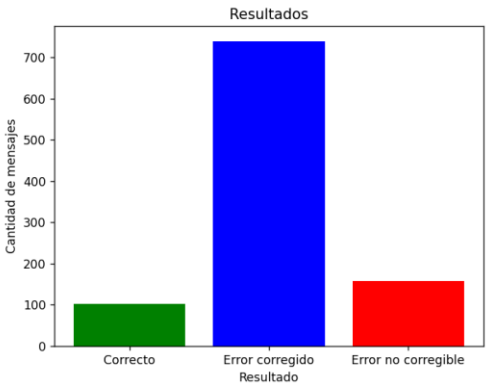


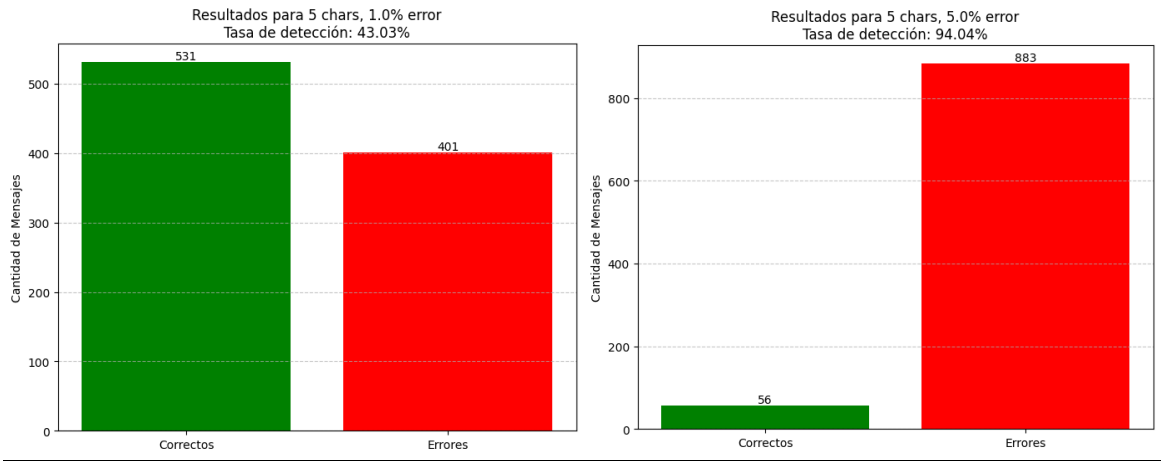
Figura 5. Longitud 15 con 5% de probabilidad de error

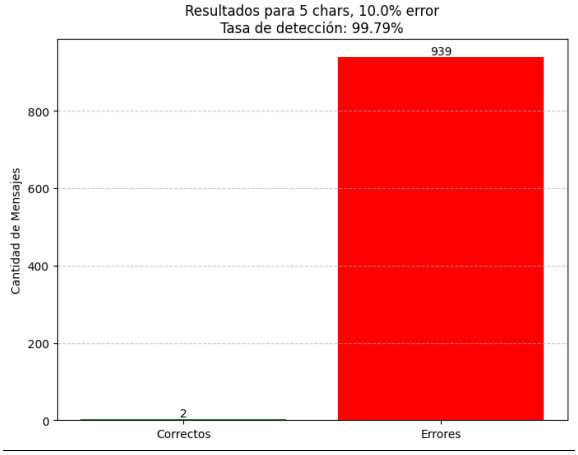
Figura 6. Longitud 15 con 20% de probabilidad de error

Algoritmo Fletcher Checksum

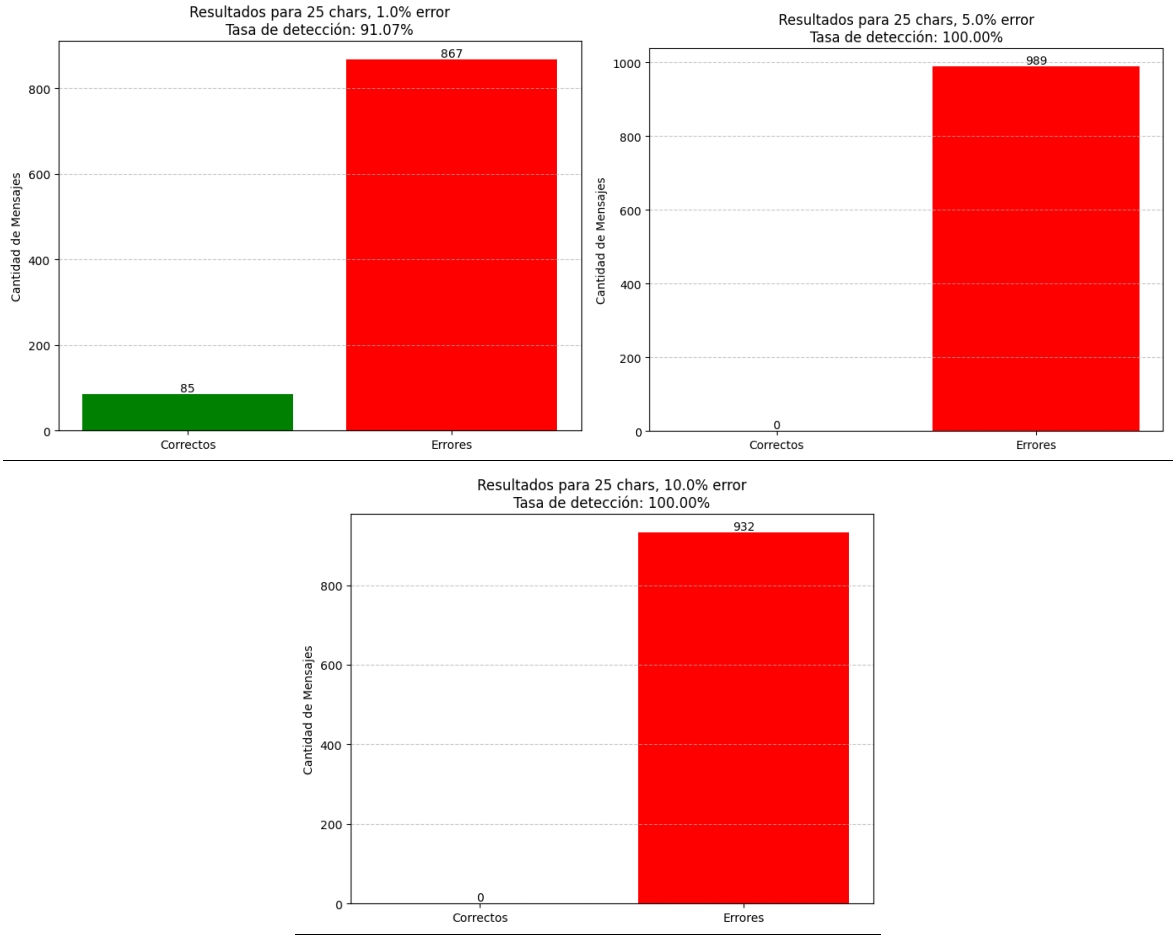
```
Resultados
Configuración  | Correctos | Errores | % Detección
-------------------------------------------------
 5.0 chars   1% |    531.0 |   401.0 |  43.03%
 5.0 chars   5% |     56.0 |   883.0 |  94.04%
 5.0 chars  10% |      2.0 |   939.0 |  99.79%
25.0 chars   1% |     85.0 |   867.0 |  91.07%
25.0 chars   5% |      0.0 |   989.0 | 100.00%
25.0 chars  10% |      0.0 |   932.0 | 100.00%
50.0 chars   1% |     15.0 |   982.0 |  98.50%
50.0 chars   5% |      0.0 |   897.0 | 100.00%
50.0 chars  10% |      0.0 |   478.0 | 100.00%

Tiempo total: 3.26 segundos
Mensajes por segundo: 2467.99
```

Figura 7. Resultados algoritmo Fletcher Checksum

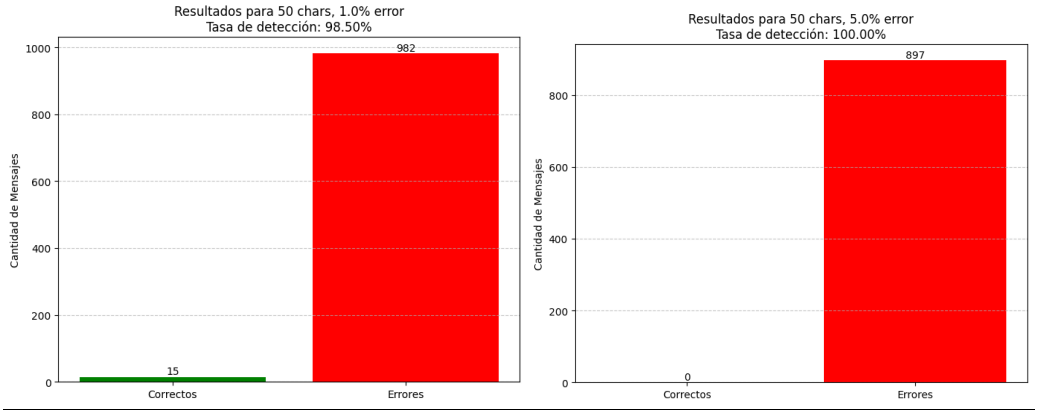


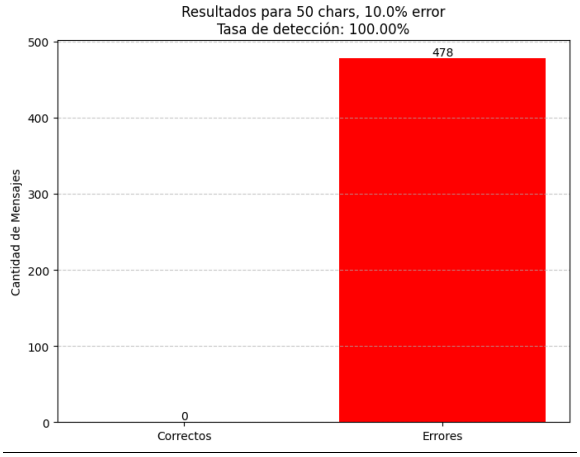


Figuras 8, 9 y 10. Longitud 5 con 1%, 5% y 10% de probabilidad de error



Figuras 11, 12 y 13. Longitud 25 con 1%, 5% y 10% de probabilidad de error





Figuras 14, 15 y 16. Longitud 50 con 1%, 5% y 10% de probabilidad de error

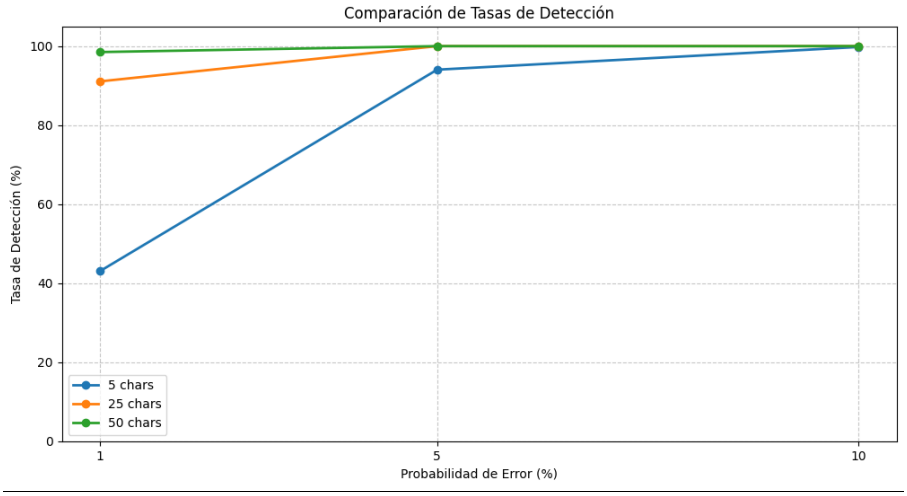


Figura 17. Comparación de Tasas de Detección

Discusión

¿Qué algoritmo tuvo un mejor funcionamiento y por qué?

El código de Hamming demostró un excelente rendimiento en la corrección de errores, especialmente en mensajes de mayor longitud, donde logró corregir la mayoría de los errores incluso con tasas de error elevadas. Esto se debe a su capacidad para identificar y corregir errores individuales gracias a los bits de paridad distribuidos estratégicamente. Sin embargo, en mensajes muy cortos, su eficiencia disminuyó, ya que la redundancia añadida puede no ser suficiente para manejar múltiples errores en tramas pequeñas. En comparación con Fletcher checksum, que solo detecta errores, Hamming sobresale en escenarios donde la corrección es crítica, pero requiere más overhead en la transmisión.

¿Qué algoritmo es más flexible para aceptar mayores tasas de errores y por qué?

El algoritmo de Hamming mostró mayor flexibilidad en entornos con altas tasas de error como 5% o 20%, ya que no solo detecta errores, sino que los corrige, manteniendo la integridad del mensaje. Esto lo hace ideal para canales muy ruidosos donde la retransmisión no es práctica. En cambio, Fletcher checksum, al ser solo de detección, depende de que el receptor solicite una retransmisión ante errores, lo que lo hace menos eficiente en condiciones de alto ruido. No obstante, Hamming tiene limitaciones con múltiples errores simultáneos, mientras que Fletcher puede detectarlos aunque sin corregirlos.

¿Cuándo es mejor utilizar un algoritmo de detección de errores en lugar de uno de corrección de errores y por qué? ¿Qué consideraciones adicionales habría que tener si optamos por 'solo' detectar errores?

Los algoritmos de detección de errores como Fletcher checksum son más adecuados en escenarios donde la eficiencia y la velocidad de transmisión son prioritarias. Por ejemplo, en redes confiables con baja tasa de errores o en protocolos que ya incluyen mecanismos de retransmisión, la detección es suficiente, ya que los paquetes corruptos pueden solicitarse nuevamente sin necesidad de agregar la sobrecarga de bits que requieren los métodos de corrección. Además, son ideales para aplicaciones donde el tamaño del mensaje debe optimizarse, como en transmisiones de datos masivos o en sistemas con ancho de banda limitado.

Ahora, al optar por solo detección, es crucial garantizar que el protocolo de comunicación incluya un método para manejar los errores detectados, como solicitudes de retransmisión. Esto puede no ser viable en entornos con alta latencia como comunicaciones satelitales o en aplicaciones en tiempo real como videollamadas, donde el retraso por retransmisión afectaría la experiencia del usuario.

Además, en canales con tasas de error muy altas, la constante retransmisión podría saturar la red, reduciendo su eficiencia. Por ello, en estos casos, un algoritmo de corrección como Hamming sería más apropiado, a pesar de su mayor overhead.

Comentarios y hallazgos

Algoritmo Hamming

Nuestra hipótesis era que mensajes de mayor longitud tendrían significativamente más errores no corregibles que aquellos de menor longitud, y que una alta probabilidad de ruido resultaría en un número considerablemente mayor de mensajes no corregibles. Sin embargo, los resultados de las pruebas comprobaron lo contrario. Como se observa en las figuras 1 a la 6, la diferencia no es notoria y, en todos los casos, los mensajes exhiben comportamientos similares, manteniéndose dentro del mismo rango de errores.

Algoritmo Fletcher Checksum

Los resultados muestran que la tasa de detección de errores varía significativamente según la longitud del mensaje y la probabilidad de error. Para mensajes cortos, el algoritmo tuvo un rendimiento inconsistente, con una tasa de detección de solo 43.03% en un entorno con 1% de error, lo que sugiere limitaciones en tramas pequeñas. Sin embargo, al aumentar la probabilidad de error, la detección mejoró notablemente, 94.04% y 99.79%. Esto muestra que el método es más efectivo en canales más ruidosos. Ahora, en mensajes más largos, el algoritmo demostró una alta confiabilidad, alcanzando tasas de detección del 91.07% al 100%, incluso con bajas probabilidades de error. Esto muestra que la redundancia añadida en tramas más grandes mejora la robustez del sistema. La gráfica comparativa confirma que, a mayor longitud del mensaje, mayor es la eficacia del algoritmo, independientemente de la tasa de error introducida. Con esto se puede respaldar la elección de algoritmos como Fletcher checksum para tramas extensas, mientras que para mensajes cortos en entornos con bajo ruido, podría requerirse ajustes o métodos complementarios.

Conclusiones

- Hamming es ideal para corrección en mensajes largos y canales ruidosos, pero pierde eficacia en tramas cortas debido a limitaciones de redundancia.
- Fletcher checksum es óptimo para detección eficiente en redes confiables, aunque su rendimiento disminuye en mensajes cortos con bajo ruido.

- La elección del algoritmo depende del contexto. Hamming para corrección en entornos críticos con alta latencia/ruido y Fletcher para detección ligera en redes estables con retransmisión. La longitud del mensaje y la tasa de error son factores clave en su efectividad.