Universidad del Valle de Guatemala

Facultad de ingeniería

Redes

Catedrático: Jorge Yass

Laboratorio #2 – Segunda parte

Esquemas de detección y corrección de errores

Nelson Eduardo García Bravatti 22434

Joaquín André Puente Grajeda 22296

Guatemala, julio de 2025

Descripción de la práctica:

En este laboratorio se profundizó en el estudio de mecanismos para manejar los errores que surgen en la transmisión de datos, uno de los retos fundamentales en el diseño de sistemas de comunicación confiables. Dado que el ruido y las interferencias son inherentes a cualquier canal de comunicación, el desarrollo de técnicas de detección y corrección de errores es esencial para garantizar la integridad de la información transmitida.

La práctica se estructuró en torno a los siguientes objetivos:

- Comprender y aplicar un modelo de capas, identificando claramente las funciones de cada nivel en el proceso de transmisión.
- Implementar la transmisión de información a través de sockets, simulando un canal real de comunicación.
- Experimentar con el envío y recepción de datos sujetos a diferentes niveles de ruido, observando el impacto de errores aleatorios.
- Analizar el desempeño y las limitaciones de distintos algoritmos de detección y corrección de errores.

El desarrollo se centró en la implementación de una aplicación modular basada en una arquitectura de capas, con las siguientes funciones principales:

- Aplicación: Solicita el mensaje a transmitir y elige el algoritmo de integridad a utilizar, mostrando al receptor el mensaje recibido o un aviso de error en caso de detectar problemas irrecuperables.
- Presentación: Codifica el mensaje original a binario ASCII para su transmisión y decodifica el mensaje recibido, devolviendo los caracteres originales si no se detectan errores.
- Enlace: Se encarga de calcular o verificar la integridad del mensaje usando el algoritmo seleccionado, concatenando la información de redundancia necesaria y, en caso de algoritmos de corrección, intentando recuperar el mensaje original si es posible.
- Ruido: Simula las interferencias en el canal aplicando errores aleatorios sobre la trama antes de su transmisión, de acuerdo con una tasa de error establecida por el usuario.
- Transmisión: Realiza el envío y recepción de mensajes mediante sockets, asegurando la comunicación entre el emisor y el receptor a través de los puertos definidos.

Algoritmos implementados:

Detección -> FletcherChecksum

Corrección -> Hamming

Resultados:

Pruebas con Fletcher Checksum:

Figura 1: Pruebas Fletcher Checksum

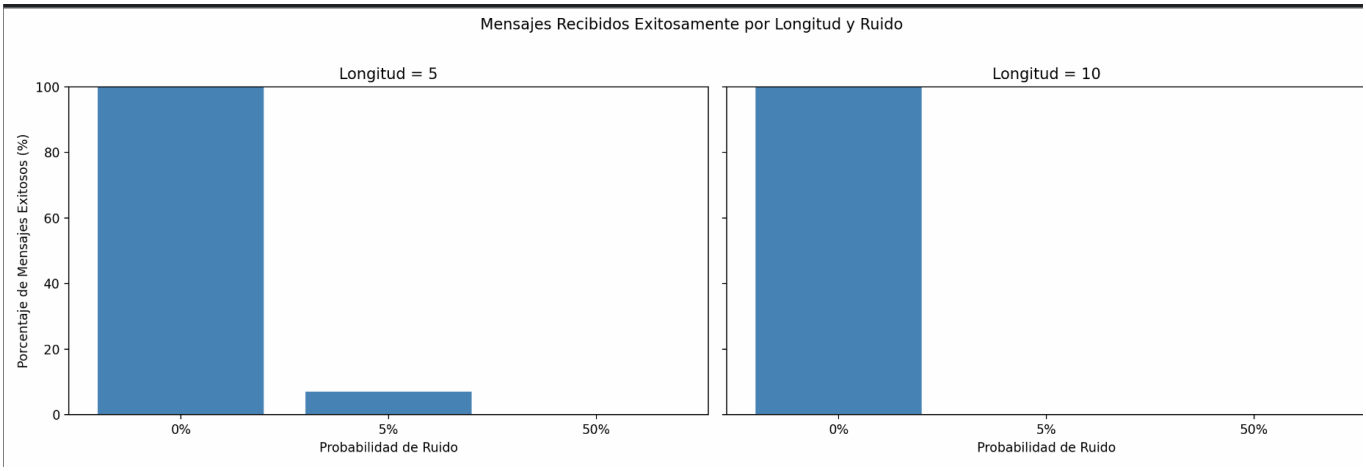
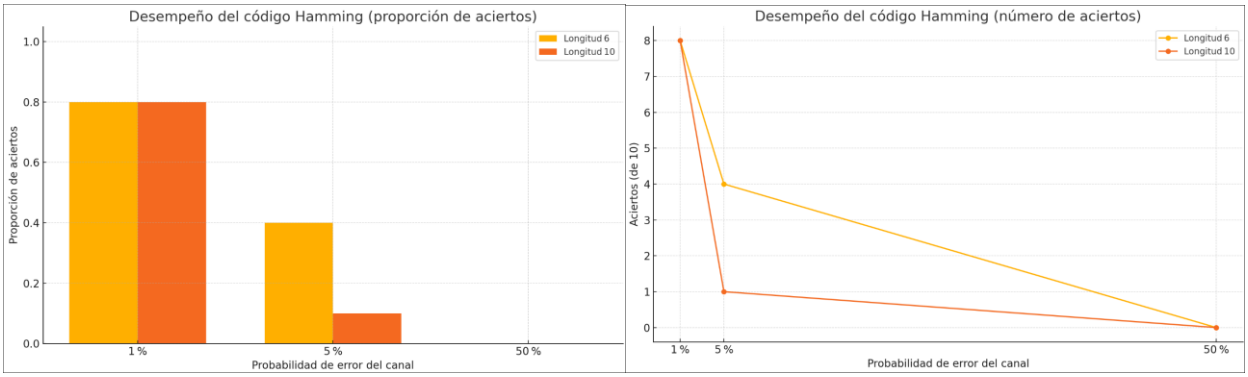


Figura 2: Pruebas de Hamming



Laboratorio 2.2 Redes UVG 2025

Discusión:

En la presente práctica se implementó el algoritmo de checksum de Fletcher-16 como mecanismo de detección de errores en la transmisión de mensajes, siguiendo una arquitectura por capas que simula una pila de protocolos realista.

El algoritmo de Fletcher-16 opera sumando acumulativamente los valores de cada bit del mensaje (en este caso, representados como '0' y '1' tras la codificación ASCII), y concatenando dos sumas parciales como la información de integridad. Su fortaleza radica en que no solo suma todos los bits, sino que también tiene en cuenta el orden, lo que lo hace más sensible a errores que el checksum tradicional de suma simple. De acuerdo con GeeksforGeeks (2024) y la documentación de Intel, el método es eficiente y rápido, permitiendo su implementación en aplicaciones que requieren bajo overhead computacional.

Los resultados experimentales (ver Figura 1) evidencian que Fletcher-16 puede detectar la mayoría de los errores cuando la probabilidad de ruido es baja o nula (por ejemplo, 0% de errores, donde la tasa de mensajes exitosos es del 100%). Sin embargo, al aumentar la longitud de los mensajes y la probabilidad de ruido, la tasa de detección de errores disminuye drásticamente: incluso con una probabilidad de error del 5%, la cantidad de mensajes recibidos sin error cae de manera significativa, y para tasas altas como 50% el éxito es prácticamente nulo.

El algoritmo no es flexible ante altas tasas de error, ya que Fletcher-16 es un esquema únicamente de detección, no de corrección. Su probabilidad de pasar por alto errores ("colisiones") aumenta conforme se incrementa el número de bits afectados por el ruido, sobre todo en mensajes largos donde errores múltiples pueden compensarse entre sí y no ser detectados (GeeksforGeeks, 2024). Además, tanto la información del mensaje como el propio checksum pueden verse afectados por el ruido, reduciendo su efectividad.

La principal utilidad de un algoritmo de detección, como Fletcher-16, está en sistemas donde los errores son poco frecuentes, o donde se prefiere descartar mensajes corruptos y solicitar retransmisión (por ejemplo, protocolos con ACK/NAK como TCP). Es decir, la detección es

preferible cuando el canal es relativamente confiable o se puede permitir el costo de volver a enviar datos. En cambio, para canales ruidosos, algoritmos de corrección (como códigos convolucionales o Reed-Solomon) son más apropiados, aunque requieren mayor complejidad y overhead.

El desempeño limitado ante altos niveles de ruido se debe tanto a la linealidad del algoritmo como a la imposibilidad de corregir errores detectados. Si bien Fletcher-16 mejora el checksum simple al incorporar dependencia de orden, sigue siendo vulnerable a combinaciones de errores que pueden pasar desapercibidas, especialmente en tramas largas o con ruido intenso (Intel; Yonex, n.d.).

Complementariamente se probó un código de Hamming (SEC) sobre cadenas de 6 y 10 bits con probabilidades de error de 1 %, 5 % y 50 %. Los resultados mostraron que, a 1 % de ruido, el decodificador corrigió correctamente 8 de 10 palabras en ambos tamaños (≈ 80 % de éxito). Al aumentar el error a 5 %, la tasa de aciertos cayó a 40 % para palabras de 6 bits y a solo 10 % para las de 10 bits; con 50 % de ruido el rendimiento fue nulo. Esto confirma la premisa teórica del código de Hamming: puede corregir únicamente un bit erróneo por palabra, por lo que su efectividad depende de que la probabilidad de ocurrencia de dos o más errores simultáneos sea despreciable. Asimismo, evidencia que los esquemas de corrección simples sufren con la longitud de la palabra y con canales moderadamente ruidosos, reforzando la necesidad de seleccionar códigos más robustos o agregar redundancia adicional cuando se espera un entorno de transmisión adverso.

Conclusiones:

La implementación del checksum de Fletcher-16 en la arquitectura de capas demostró ser adecuada para detectar errores simples en canales con bajo nivel de interferencia, gracias a su eficiencia y simplicidad. Sin embargo, no es suficiente para entornos con altos niveles de ruido, donde se requiere mayor robustez. Los resultados experimentales respaldan la teoría, mostrando una clara caída en el porcentaje de mensajes exitosos conforme aumenta la probabilidad de error y la longitud del mensaje.

Al complementar la prueba con un **código de Hamming de corrección de un bit**, se observó que la capacidad de reconstruir el mensaje mejora en canales ligeramente ruidosos (≈ 80 % de éxito con 1 % de error), pero sigue deteriorándose rápidamente cuando la probabilidad de tener dos o más errores en una misma palabra deja de ser despreciable (40 %–10 % de éxito con 5 % de error y 0 % con 50 %). Además, la eficacia disminuye a medida que crece la longitud de la palabra, pues aumentan las combinaciones posibles de errores múltiples que sobrepasan la capacidad correctiva.

En conjunto, los ensayos confirman que:

1. **Fletcher-16** es idóneo como esquema de **detección** cuando el canal es confiable y se puede retransmitir.
2. **Hamming (SEC)** aporta **corrección** básica, adecuada solo cuando la tasa de error media es < 1 % y las palabras son relativamente cortas.
3. Para canales moderados o severamente ruidosos es imprescindible incorporar códigos más potentes (p. ej., SECDED, BCH, Reed-Solomon) o técnicas híbridas que combinen detección y corrección junto con mecanismos de retransmisión, asumiendo el mayor overhead que ello implica.

Bibliografía:

GeeksforGeeks. (2024, May 28). Understanding checksum Algorithm for data integrity. GeeksforGeeks. https://www.geeksforgeeks.org/system-design/understanding-checksum-algorithm-for-data-integrity/

Fast computation of Fletcher checksums. (n.d.). Intel. https://www.intel.com/content/www/us/en/developer/articles/technical/fast-computation-of-fletcher-checksums.html

Yonex. (n.d.). Fletcher. Scribd. https://www.scribd.com/document/154048170/Fletcher