Universidad del Valle de Guatemala
CC3067 - Redes
Adrian Rodríguez 21691
Samuel Chamalé 21881

Laboratorio #2 - Esquemas de detección y corrección de errores
Segunda Parte

Repositorio
https://github.com/chamale-rac/detection-correction/

Video de Demostración (TCP - Sockets)
https://youtu.be/Y64_UqS_8xE?si=0NYzGvkXkIk_Z5P-

Lenguajes utilizados

- Rust: Emisor
- Go: Receptor

Descripción
En esta segunda parte del laboratorio, pusimos en práctica los algoritmos de detección y corrección seleccionados en la primera parte (CRC32 y Hamming). Transmitimos y recibimos mensajes a través de diversas capas y servicios, con el objetivo de experimentar un flujo real en un canal de comunicación poco confiable, caracterizado por interferencias y ruido. Además, con el objetivo de analizar la precisión de dichos algoritmos, se generaron casos de prueba del lado del emisor, para posteriormente evaluar a los dos algoritmos.

Metodología
Los casos de prueba fueron generados del lado del emisor, utilizando 15 palabras de complejidad creciente y variando la tasa de error. Estos casos fueron guardados en un archivo CSV, con los siguientes atributos:

- **Error Rate**: La tasa de error introducida en el canal de comunicación.
- **Length**: La longitud de la palabra original.
- **Original**: La palabra original antes de ser codificada.
- **Encoded**: La representación binaria de la palabra original después de ser codificada.
- **Noisy**: La representación binaria después de la introducción de errores.
- **Errors**: El número de errores detectados en la representación binaria ruidosa.

A continuación se evidencia una muestra de los casos de prueba:

	Error Rate	Length	Original	Encoded	Noisy	Errors
0	0.1	5	apple	1100110110100100011110000000000111100000001100...	1000110110100100011110000000100111100000001100...	7
1	0.1	6	banana	1100110010101100110110100111001100010110110 0...	1101110010100011001101111001110001011101101100...	11
2	0.1	6	cherry	1100110100001111001101110000110011001001010001...	1100110100001110001101110000110011000001010001...	3
3	0.1	6	grapes	1100110000111100011110101010110011011010010001...	1100110000110010011110001010110011011010010001...	7
4	0.1	9	kiwifruit	1100110011001111001100011001000111100011111100...	0101110011001101001100011001000111100110110100...	17

Posteriormente, estos casos de prueba fueron leídos en el receptor utilizando Go. Se implementó un proceso para decodificar los mensajes utilizando el algoritmo de Hamming (7,4) y comparar los mensajes decodificados con los originales. Los resultados de este proceso se guardaron en otro archivo CSV con los siguientes atributos:

- **Error Rate**: La tasa de error introducida en el canal de comunicación.
- **Length**: La longitud de la palabra original.
- **Errors**: El número de errores en el mensaje decodificado.
- **Precision**: La precisión del mensaje decodificado.
- **Detected Errors**: El número de errores detectados por el código Hamming.
- **Original Message**: El mensaje original antes de la codificación.
- **Decoded Message**: El mensaje después de ser decodificado.

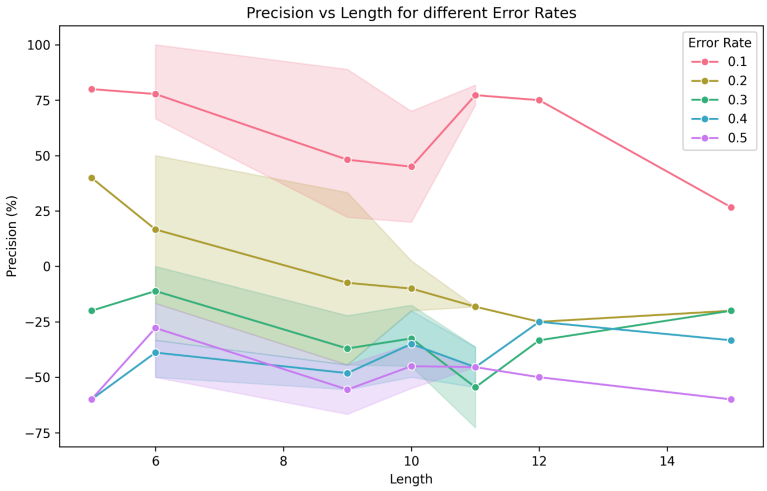
La precisión se calculó de la siguiente manera:

$$P = 1 - \frac{N}{L}$$

Donde **N** es la cantidad de errores (diferencias entre el mensaje original y el decodificado), y ***L*** la longitud del mensaje original.

Discusión

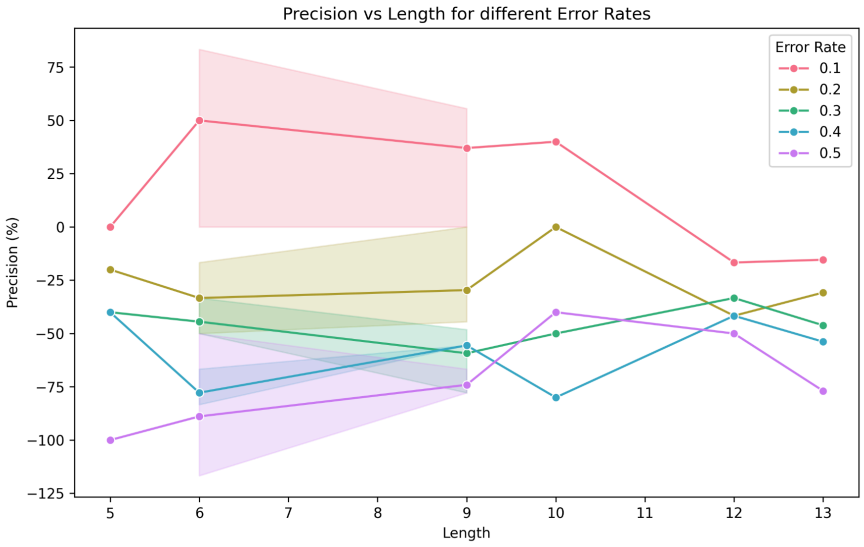
Hamming 7,4



Se observa que, a medida que la tasa de error aumenta, la precisión disminuye uniformemente. Para una tasa de error de 0.1, la precisión comienza alta pero disminuye con el aumento de la longitud del mensaje, recuperándose ligeramente en longitudes mayores. En tasas de error más altas (0.4 y 0.5), la precisión es consistentemente baja y negativa, indicando una alta cantidad de errores en la decodificación.

El ruido y los errores se distribuyen de manera uniforme, afectando de manera similar todas las longitudes de mensajes, aunque los efectos son más pronunciados en mensajes más largos para tasas de error más altas. Esto sugiere que, mientras Hamming (7,4) es eficaz en condiciones de bajo error, su desempeño se degrada significativamente en entornos con altas tasas de error.

Hamming 15,11



Para una tasa de error de 0.1, la precisión comienza alta, alcanzando un máximo cercano al 75%, y aunque hay fluctuaciones, la precisión se mantiene relativamente alta en comparación con otras tasas de error. En tasas de error más altas (0.4 y 0.5), la precisión es significativamente baja, con valores que incluso pueden ser negativos, lo que indica una alta cantidad de errores en la decodificación.

Se puede notar que la longitud del mensaje influye en la precisión, especialmente en tasas de error más altas. Sin embargo, para mensajes más largos, hay puntos donde la precisión mejora, sugiriendo que Hamming (15,11) podría ser más robusto en ciertas condiciones en comparación con Hamming (7,4). En general, el desempeño se degrada con tasas de error más altas, pero muestra una capacidad de recuperación en longitudes específicas, lo que indica una complejidad adicional en la relación entre la longitud del mensaje y la precisión en la presencia de ruido uniforme.

CRC32

En el caso de CRC32, no hay mucho que se pueda graficar, el algoritmo solo detecta errores, pero no los corrige. La detección de errores por parte de CRC32 es muy eficiente debido al tamaño del polinomio utilizado, lo que hace que sea muy difícil que un error pase desapercibido. De hecho, la probabilidad de que un error no sea detectado es aproximadamente $\frac{1}{2^{32}}$ (Prieto , 2015).

¿Qué algoritmo tuvo un mejor funcionamiento?

El rendimiento de CRC32 y Hamming no se puede comparar directamente en términos de precisión o capacidad de corrección, ya que CRC32 solo detecta errores, mientras que Hamming corrige errores. Sin embargo, en términos de detección de errores, CRC32 es extremadamente eficiente debido al tamaño de su polinomio, lo que hace que sea muy difícil que un error pase desapercibido.

Por otro lado, Hamming (7,4) y Hamming (15,11) no solo detectan errores sino que también pueden corregir uno o más errores en un bloque de datos. En un entorno con ruido moderado, Hamming (15,11) mostró una mejor precisión y capacidad de recuperación en ciertas longitudes de mensajes, lo que sugiere una mayor robustez en condiciones específicas.

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

Hamming (15,11) demostró ser más flexible en términos de aceptar mayores tasas de errores en comparación con Hamming (7,4), debido a su mayor longitud de código, lo que le permite manejar y corregir más errores. Sin embargo, ambos algoritmos de Hamming tienen un límite en la cantidad de errores que pueden corregir. CRC32, al ser solo un detector, puede manejar cualquier tasa de error pero no proporciona mecanismos para corregir esos errores.

¿Cuándo es mejor utilizar un algoritmo de detección de errores en lugar de uno de corrección de errores?

- Detección de errores (CRC32):
    - Es más adecuado en sistemas donde la transmisión es confiable y la probabilidad de errores es baja.
    - Ideal para situaciones donde el costo de retransmitir datos es bajo.
    - Utilizado comúnmente en redes y protocolos de comunicación para verificar la integridad de los datos.
- Corrección de errores (Hamming):
    - Es mejor en entornos con alta probabilidad de errores, donde la corrección en tiempo real es crucial.
    - Adecuado para sistemas donde la retransmisión no es viable o es costosa.
    - Utilizado en almacenamiento de datos, comunicaciones satelitales y otros sistemas críticos donde la corrección de errores es esencial.

Comentarios

Después de haber explorado, aunque de forma introductoria, el tema de los algoritmos de codificación y detección, consideramos que este campo es sumamente interesante y relevante. Los métodos que estudiamos además de sofisticación, muestran creatividad en su diseño y aplicación. Es interesante como estos algoritmos tienen gran relevancia para garantizar la integridad y seguridad en múltiples contextos. Será interesante observar qué avances habrán en este campo y cómo esto impactará en la seguridad informática.

La restricción de utilizar un lenguaje para el emisor y el receptor me pareció bastante creativa, y fue satisfactorio lograr comunicar los mensajes a través de TCP. Utilizar Rust fue algo complejo y en retrospectiva hubiéramos podido utilizar un lenguaje más sencillo, sobre todo cuando lo importante es aprender conceptos.

Conclusiones

La transmisión de mensajes se realizó a través de una comunicación TCP entre un cliente en Rust y un servidor en Go, simulando un canal de comunicación poco confiable con la introducción de ruido. El ruido afecta uniformemente todas las longitudes de mensajes, lo que permitió evaluar la robustez de los algoritmos bajo diferentes tasas de error.

CRC32 demostró ser bastante eficiente en la detección de errores, sin embargo, su incapacidad para corregir errores limita su aplicación en entornos con alta tasa de errores. Hamming (7,4) y Hamming (15,11) no solo detectaron errores sino que también los corrigieron. Hamming (15,11) mostró una mejor precisión y capacidad de recuperación en ciertas longitudes de mensajes, lo que sugiere una mayor robustez en condiciones específicas y una mayor flexibilidad para aceptar mayores tasas de error en comparación con Hamming (7,4).

Referencias

https://www.ieee802.org/3/bv/public/Jan_2015/perezaranda_3bv_2_0115.pdf