Universidad del Valle de Guatemala
Redes - Sección 10
CC3067

Laboratorio 2

Esquemas de detección y corrección de errores

Link de Github: https://github.com/martspain/ErrorDetectionLab_Networking

Martín España	19258
Alejandra Gudiel	19232
Laura Tamath	19365

Antecedentes

Errores suceden en toda comunicación, y es parte de los retos al momento de implementar este tipo de sistemas el manejar adecuadamente las fallas que puedan ocurrir. Por lo tanto, a lo largo de la evolución del Internet se han desarrollado distintos mecanismos que sirven tanto para la detección como para la corrección de errores.

Descripción de la práctica

En este laboratorio se identificarán las ventajas y desventajas para la detección de errores que surgen en la comunicación, ya que permiten conocer si hubo o no un error en la cadena de datos recibidos. Además se comprenderán los elementos de una arquitectura en capas.

Dicho modelo se conforma de las siguientes capas:

Emisor	**Receptor**
1. Aplicación 2. Verificación 3. Ruido 4. Transferencia	1. Transmisión 2. Codificación 3. Verificación 4. Aplicación

Los algoritmos utilizados son los siguientes:

- Código de Hamming

  Un código de Hamming es un código lineal para la detección de errores que puede detectar hasta dos errores de bits simultáneos y es capaz de corregir errores de un solo bit. Se garantiza una comunicación confiable si la distancia entre el transmisor y el receptor es menor o igual a uno.

  Se basa en que cada bloque de cinco bits (conocido como penta-bit) tuviera exactamente dos unos, asegurando así que tenga una distancia de Hamming igual a dos. De este modo, la computadora podría detectar posibles errores cuando en su entrada no había exactamente dos unos en cada penta-bit.

- Fletcher checksum

  Este algoritmo permite implementar una suma de comprobación la cual ayuda a detectar un error en alguno de los campos de un mensaje entre su emisión y recepción.

  En términos generales, este divide los datos binarios que se va a proteger de errores en "bloques" cortos de bits y calcula la suma modular de esos bloques. Un ejemplo puede ser que se necesitan transmitir 136 caracteres, cada uno de estos está almacenando como un byte de 8 bits, formándose así una palabra de datos de 1088 bits en total, con módulo conveniente de 255. Entonces, la suma de verificación simple se calcula sumando todos los bytes de 8 bits del mensaje, dividiéndolos por 255 y manteniendo solo el resto. El valor de la suma de comprobación se transmite con el mensaje, aumentando su longitud a 137 bytes o 1096 bits. El receptor del mensaje puede volver a calcular la suma de verificación y compararla con el valor recibido para determinar si el mensaje ha sido alterado por el proceso de transmisión.

- CRC-32

Una verificación de redundancia cíclica (CRC) es un código de detección de errores comúnmente utilizado en redes digitales y dispositivos de almacenamiento para detectar cambios accidentales en datos digitales. Este es un algoritmo que detecta cambios entre datos origen y destino. Los CRC se llaman así porque el valor de verificación (verificación de datos) es una redundancia (amplía el mensaje sin agregar información) y el algoritmo se basa en códigos cíclicos. Los CRC son populares porque son fáciles de implementar en hardware binario, fáciles de analizar matemáticamente y particularmente buenos para detectar errores comunes causados por el ruido en los canales de transmisión.

Además, también es una función conocida como CRC32 convierte una cadena de longitud variable en una cadena de 8 caracteres que es una representación textual del valor hexadecimal de una secuencia binaria de 32 bits. La función CRC32 devuelve una cadena de 8 caracteres que es una representación textual del valor hexadecimal de una secuencia binaria de 32 bits. La función CRC32 de Amazon Redshift se basa en el polinomio CRC-32C.

A continuación se detallan los resultados obtenidos de la práctica usando los algoritmos previamente mencionados.

Resultados:

Tabla 1: porcentaje de error en Fletcher			
No. intento	1%	10%	50%
1	0	0.025	0.1
2	0	0.001	0.24
3	0	0.047	0.087
4	0	0.034	0.099
5	0	0.0044	0.32
6	0	0.0009	0.28

Cuadro no. 1: Porcentaje de error en Fletcher

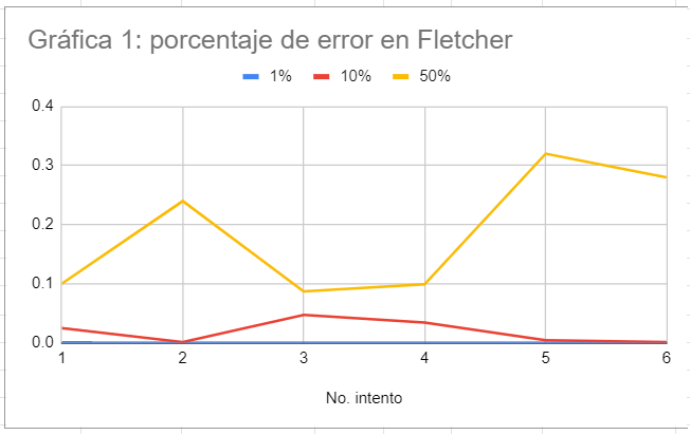

Figura no. 1: Porcentaje de error en Fletcher

Tabla 2: porcentaje de error en CRC-32			
No. intento	1%	10%	50%
1	0	0.0035	0.69
2	0	0.0018	0.92
3	0	0.0017	0.87
4	0	0.0019	0.89
5	0	0.0002	0.96
6	0	0.0051	0.14

Cuadro no. 2: Porcentaje de error en CRC-32

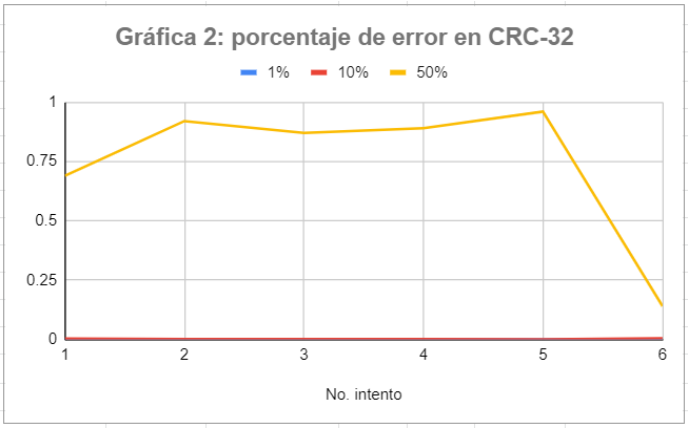

Figura no. 2: Porcentaje de error en CRC-32

Tabla 3: porcentaje de error en Hamming			
No. intento	1%	10%	50%
1	0	0.04	0.45
2	0	0.06	0.32
3	0	0.02	0.77
4	0	0.07	0.23
5	0.013	0.07	0.35
6	0	0.01	0.56

Cuadro no. 3: Porcentaje de error en Hamming

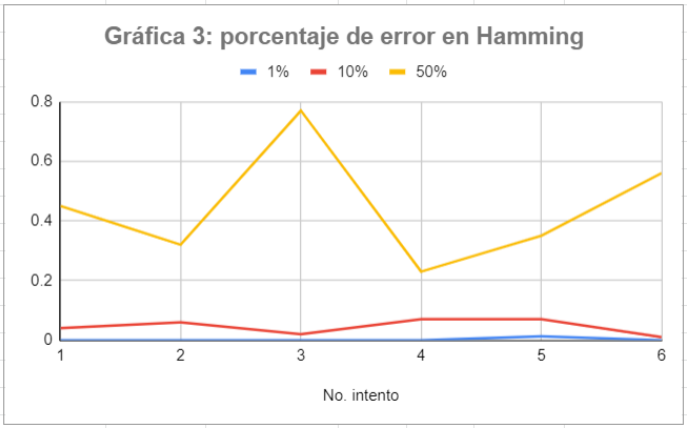

Figura no. 3: Porcentaje de error en Hamming

Discusión:

Como se puede apreciar en los resultados (véase Cuadro 1, Cuadro 2 y Cuadro 3) los porcentajes de error más altos se encuentran en el algoritmo de Fletcher y en el algoritmo de Hamming. Esto se debe a que el algoritmo de Fletcher solamente es de detección de errores, por lo que está mostrando el porcentaje real de error, siendo este bastante congruente con la probabilidad de fallo inducida al algoritmo de generación de ruido del mensaje. En cuanto al algoritmo de Hamming, puede deberse a alguna operación realizada durante la corrección de errores. De cualquier manera, en este caso el algoritmo que obtuvo un mejor rendimiento en cuanto al manejo de errores dado su porcentaje de error es el algoritmo CRC-32. Esto se puede observar más fácilmente en los gráficos obtenidos a partir de los resultados (véase Figura 1, Figura 2 y Figura 3).

En cuanto a la flexibilidad respecto de las tasas de error, el algoritmo Fletcher realmente resalta, pues al no contar con un procedimiento de corrección de errores, le es posible ignorar en cierto punto los errores mínimos y aún mandar la mayor cantidad de la información correcta. De manera evidente, esto depende directamente de la probabilidad de error inducida en la capa de ruido, lo cual en un caso de aplicación del mundo real, puede variar dependiendo de múltiples factores inherentes a la red donde se realiza la comunicación. Además, un error puede propagarse a través de las distintas capas, lo cual puede incrementar o empeorar la calidad de la información en caso se realicen procedimientos erróneos con la data.

Por lo tanto, existen diversas situaciones bajo las cuales resulta pertinente afirmar cuál algoritmo es el más conveniente para utilizar, de modo que si no es muy relevante, dado el contexto, que la información recibida tenga cierto porcentaje mínimo de error, es posible solo utilizar un algoritmo de detección de errores para determinar qué tan grave es el error, o si existe un error para empezar.

En cambio, para situaciones donde se requiere mandar una gran cantidad de información que es propensa a errores, por ejemplo una transmisión de video en vivo, puede resultar útil un algoritmo de detección y corrección de errores. Esto principalmente es debido a que si bien la corrección no es perfecta, se puede tener un resultado bastante satisfactorio del lado del recipiente. En el caso propuesto sobre una transmisión en vivo, esto podría brindar la data faltante del video o el audio enviados de modo que si ocurre una pérdida de paquetes, no sea necesario volver a enviar la información para que los mensajes sean totalmente comprensibles por sus recipientes.

Conclusiones

- Luego de realizar diferentes corridas con los algoritmos, logramos concluir que Hamming es un algoritmo más enfocado para una corrida con una longitud determinada de datos; mientras que CRC-32 y Fletcher dan mejores resultados sin importar la longitud de los datos por ingresar. Con esta reflexión, determinamos que cumplimos con el objetivo de identificar las ventajas y desventajas de implementación para la detección y corrección de errores.
- Al momento de iniciar con este laboratorio, logramos cumplir con el objetivo de comprender los elementos de una arquitectura de capas ya que todos los elementos trabajados en el código fueron seleccionados y analizados para poder cumplir con el modelo propuesto.
- Por medio del estudio y el uso de los distintos algoritmos de corrección y detección errores se puede detectar las irregularidades que a veces se presentan a la hora de transmitir datos, esto es de vital importancia ya que al detectar un error, se puede implementar una técnica de corrección acorde al error que haya ocurrido, y así poder mantener una transmisión segura de los datos.

Referencias:

- Amazon. Función CRC32. https://docs.aws.amazon.com/es_es/redshift/latest/dg/crc32-function.html
- Chandu J., (2020). Tutorials point. Fletcher's checksum: https://www.tutorialspoint.com/fletcher-s-checksum#:~:text=Fletcher%20checksum%20is%20an%20error,at%20Lawrence%20Livermore%20Labs%2C%20USA.
- Fletcher, J. G., "An Arithmetic Checksum for Serial Transmissions", IEEE Trans. on Comm., Vol. COM-30, No. 1, January 1982, pp 247-252. (https://en-academic.com/dic.nsf/enwiki/368961)
- Informatica (2022) CRC32 https://docs.informatica.com/es_es/data-integration/data-services/10-1/referencia-del-lenguaje-de-transformacion/funciones/crc32.html
- Technopedia (2021) https://es.theastrologypage.com/hamming-code