Universidad del Valle De Guatemala

Facultad de Ingeniería

Redes

Laboratorio 2.2: Esquemas de detección y corrección de errores

Javier Mombiela 20067
Javier Valle 20159

Guatemala, 10 de agosto 2023

Descripción

A lo largo de la presente práctica se usó como base la construcción del laboratorio 2.1, para poder construir un emisor y un receptor de tramas de bits. Para poder continuar el laboratorio, se modificó la entrada que recibe el receptor para que le llegara un string para luego convertirlo a un ascii binario. Asimismo, se construyó una especie de emisor de ruido con el fin de poder perturbar un poco los mensajes que se reciben. Para el ruido se hicieron unas fórmulas para poder moderar la cantidad de errores que había por string, esta fórmula se hizo en base al largo del string, ya en ascii. Luego, se creó un socket para poder crear una comunicación entre los lenguajes de programación, o sea el emisor y el receptor. El receptor debia de recibir el mensaje y ver si era posible corregirlo o no en base al ruido que se le fue agregado. Como último paso se generaron diez mil datos al azar con diferentes tamaños de string, comprendidos entre 1 a 16 caracteres, para que estos tuvieran un tamaño máximo de 128 bits, y así comparar el comportamiento de los algoritmos y los resultados que estos generan.

Resultados

Para hacer las pruebas, se generaron 10,000 datos aleatorios con largo de 1-16 caracteres. Se hicieron 625 datos aleatorios con cada uno de los tamaños. Estos son los largos ingresados, pero a la hora de convertirse a ascii binario, el largo se multiplicó por 8, por lo que en realidad el código se realiza con palabras entre tamaño 8-120. La probabilidad en cada uno de los algoritmos esta calculada en base al largo de la cadena, el punto es que con Hamming las cadenas tuvieran entre 0-2 errores y con CheckSum que tuvieran entre 0-1 error. Se realizaron dos pruebas, en la primera se calcula la probabilidad base y se le aplica un ajuste y en la segunda prueba, se elimina este ajuste. El ajuste hace que la probabilidad base se incline más hacia un valor que pueda provocar o no más errores.

Hamming:

> *probBase = 1 / largo de la cadena / 2*
> *ajuste = probBase / 2*

Checksum:

> *probBase = 1/ largo de la cadena*
> *ajuste = probBase / 2*

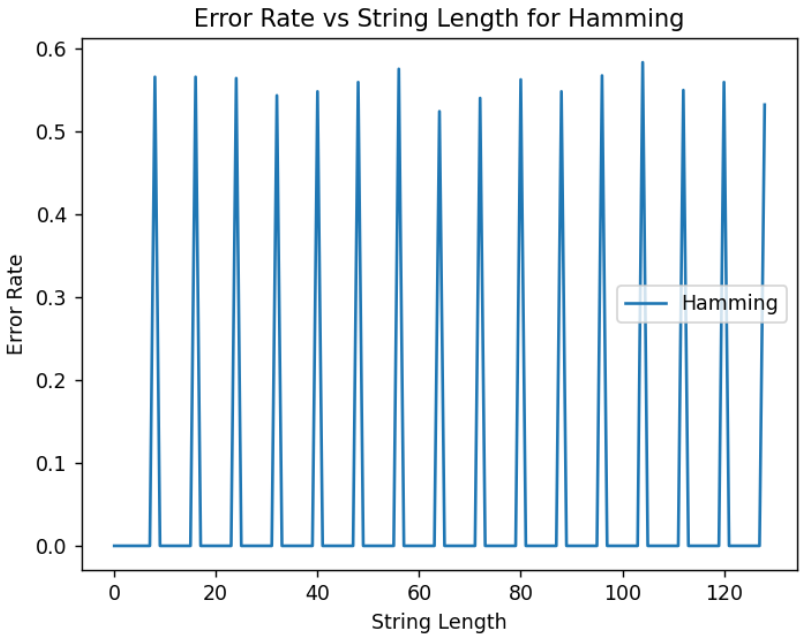
Aplicar ajuste:

valorRandom = (valor entre 0 y 1)
si valorRandom > 0.3 return probBase - ajuste

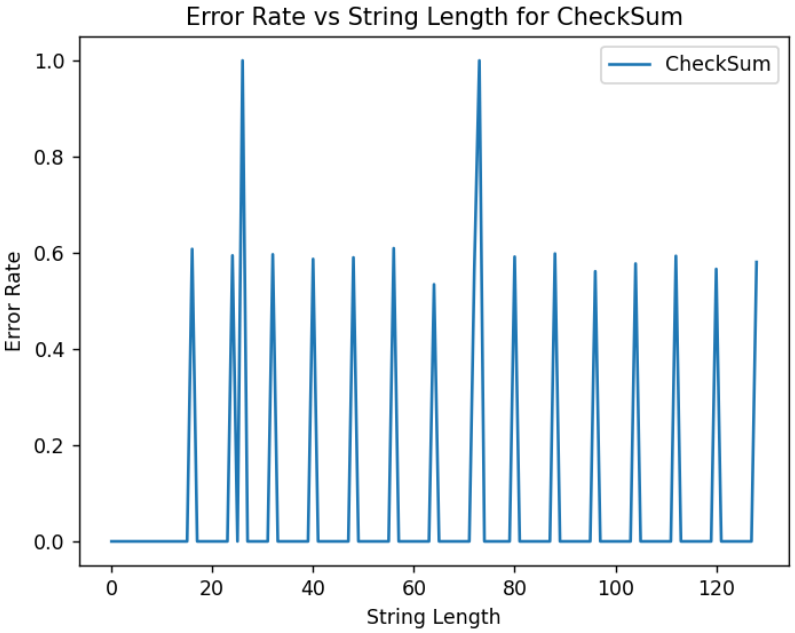
si 0.3< valorRandom < 0.6 return probBase + ajuste
else return probBae

Prueba 1: 10k datos con ruido ajustado

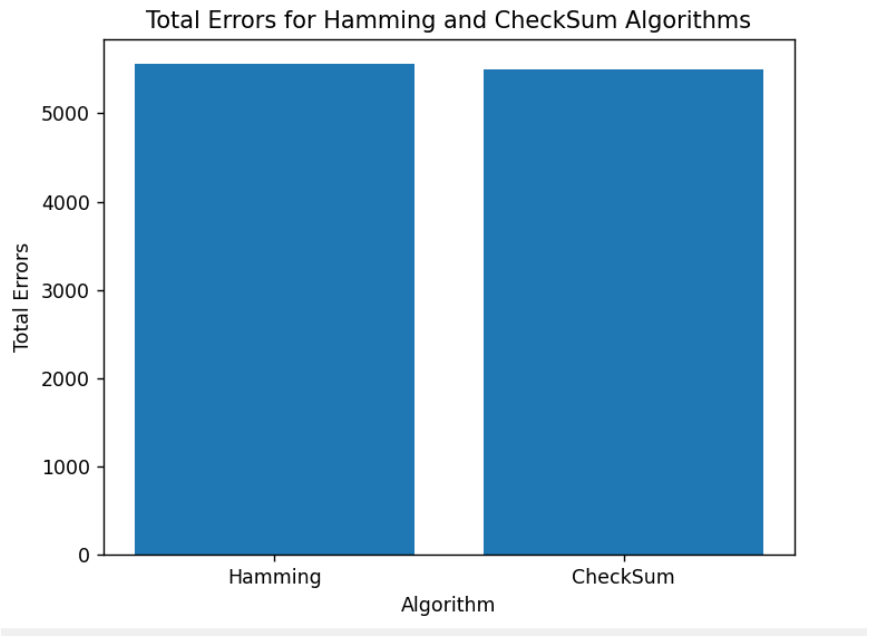
Gráfica 1: Error vs. longitud del string en Hamming



Gráfica 2: Error vs. longitud del string en Chacksum

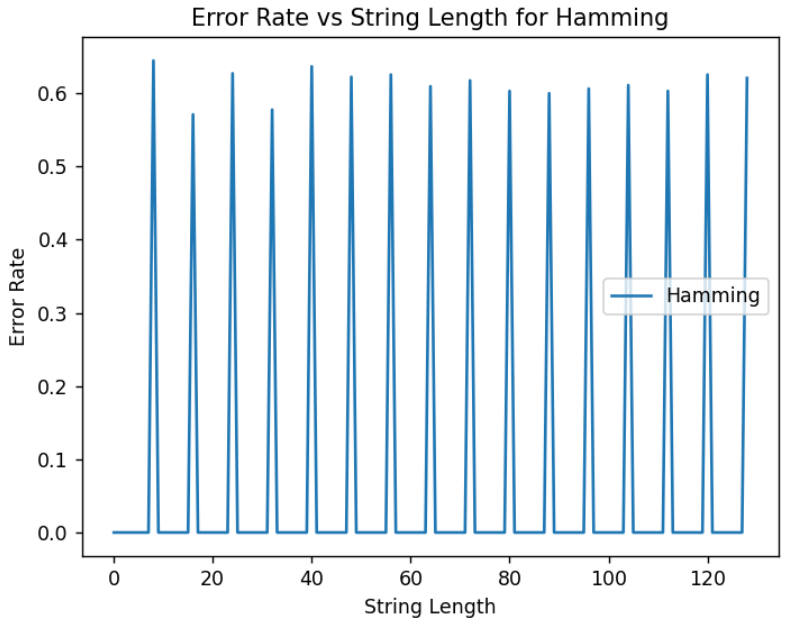


Gráfica 3: Cantidad de errores en Hamming vs. cantidad de errores en Checksum

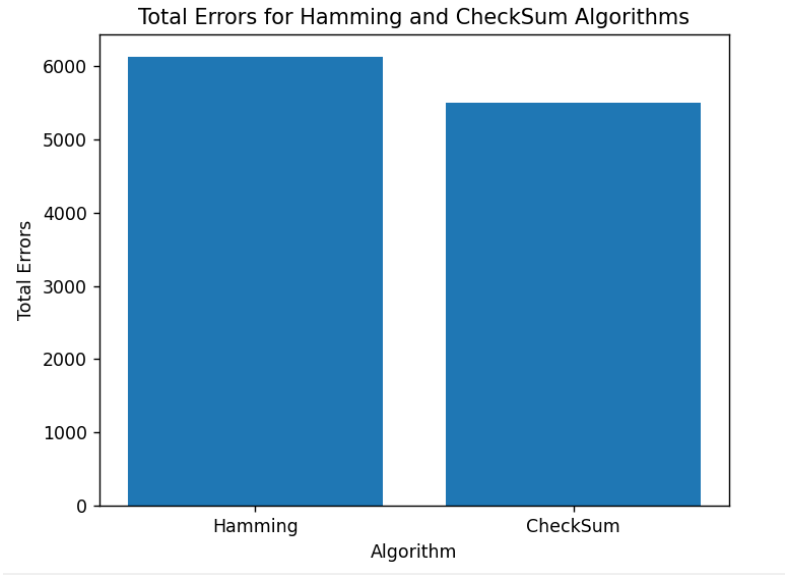


Prueba 2: 10k datos sin ajustar el ruido

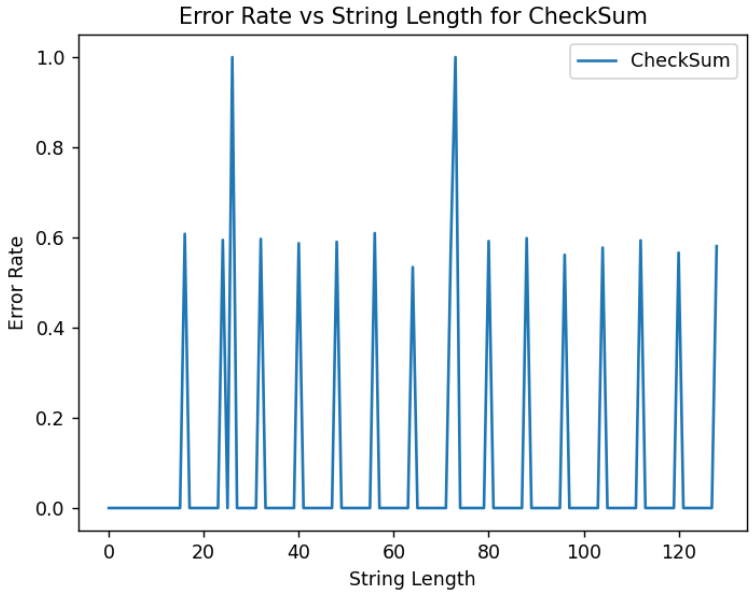
Gráfica 4: Error vs. longitud del string en Hamming



Gráfica 4: Cantidad de errores en Hamming vs. cantidad de errores en Checksum



Gráfica 6: Error vs. longitud del string en Chacksum



Discusión

A la hora de realizar las pruebas con los 10,000 datos generados aleatoriamente, logramos observar que tenemos resultados esperados, ya que los ruidos se calculan como cierto porcentaje del largo de la cadena ingresada. Para ambos algoritmos se calcularon diferentes fórmulas y esto se hizo con el objetivo de que las cadenas de CheckSum tuvieran menos bits volteados que las cadenas de Hamming. La razón por la cual se decidió hacer esto, fue debido a que CheckSum solo detecta los errores, mientas que Hamming logra corregir 1 error. Por lo tanto, con CheckSum, con uno o más errores, este ya no es capaz de decodificar el mensaje.

Por el otro lado, Hamming sí es capaz de seguir corrigiendo si este encuentra un error, pero con más de uno, falla. Para ambos se calculó una probabilidad de ruido de al rededor de 50%.

En el algoritmo de Checksum se logró notar que la cantidad de errores sí era la esperada ya que la probabilidad de encontrar cero o un error en un string era bastante alta o lo mismo que 50% y se puso esta cantidad, dado que este algoritmo solo busca encontrar errores, más no se esperaba solucionar ningún error encontrado en los strings. Por lo tanto, si la trama tenía 0 errores, lograba regresar el mensaje pero con otro error ya no. Por esta razón vemos un 50-50.

En cuanto al algoritmo de Hamming, se pudo notar que éste logró dar los resultados esperados, dado que con éste se pudo encontrar la cantidad de errores dentro de las tramas y corregir los errores en las tramas en base a la longitud de bits configurados para este código. Asimismo, en los errores que ya no estaban al alcance, este algoritmo solo notificó la trama con errores. Con Hamming el ruido se calculó para que el número de bits que se cambiaran fueran entre 0-3 bits, ya que con 0 y 1 Hamming todavía lograba corregir estos errores, pero con más ya no, por lo también vemos que el gráfico que muestra 5000 errores también está bien.

Por lo tanto, el algoritmo que mejor rendimiento tuvo dentro de la ejecución fue el de checksum, dado que este logró encontrar la cantidad de errores esperados en base a la probabilidad configurada en este. Como se mencionó anteriormente, Hamming también tuvo el comportamiento esperado en la primera prueba, pero en la segunda tuvo casi 6,000 errores.

El algoritmo que tiene más flexibilidad para aceptar una mayor tasa de errores es el de Hamming, dado que a éste se le puede configurar una probabilidad más alta de detección de errores y, además, no se le está pidiendo como tal que corrija todos los errores encontrados dentro de las tramas erróneas. Como se mencionó anteriormente, la probabilidad de ruido para el algoritmo de Hamming se calculó para que fuera mayor que esa de CheckSum.

Es mejor usar un algoritmo de detección de errores cuando no es tan relevante corregir errores en las tramas que se están enviando/recibiendo. Un algoritmo de detección también nos ayuda cuando necesitamos utilizar menos recursos. Sin embargo, cuando se está trabajando en canales o medios más vulnerables, es mejor usar un algoritmo de corrección de errores, dado que hay una necesidad más fuerte de encriptar y desencriptar con más detalle las tramas que se están enviando/recibiendo.

Otro punto importante a mencionar es que el largo de la cadena no importaba mucho en cuanto a la probabilidad de los errores, puesto que al hacer el cálculo de esta probabilidad, se logró notar que siempre se llegaba al mismo número de probabilidad de que la cadena estuviera o no errónea, y este siempre depende del largo. Por esto es que en las gráficas podemos observar que no hay diferencia entre el largo de la cadena y los errores, por la probabilidad. Por esto también es que CheckSum tuvo mejores resultados. De haber intentado con probabilidades alta, media y baja, se hubiera podido notar que obviamente el nivel de

errores incrementa cuando la probabilidad aumenta y también se hubiera podido notar que CheckSum hubiera tenido más errores que Hamming. Pero ya que estos eran los resultados esperados, decidimos hacerlo con probabilidades para ver el funcionamiento con probabilidades parejas.

Comentario Grupal

El presente laboratorio nos fue de bastante utilidad para poder visualizar como es que trabajan de manera conjunta un emisor y un receptor de tramas de bits. Asimismo, nos pareció bastante interesante ver como es el receptor presenta una gran eficiencia al momento de detectar los errores que encuentra en las tramas y como es que este mismo las corrige en base a los estándares que se le dan. También nos pareció sorprendente como es que el ruido llega a interferir en la comunicación entre el emisor y receptor; y dado este ruido como es que el receptor puede llegar a detectar también los errores en base al ruido en base a su algoritmo de decodificación.

Por otro lado, pudimos aprender sobre la importancia de la comunicación entre el emisor y el receptor para garantizar la transmisión correcta de información. También pudimos observar cómo la detección y corrección de errores son fundamentales para garantizar la integridad de los datos transmitidos.

Conclusiones

1. El algoritmo de Checksum resultó ser bastante efectivo para poder encontrar errores dentro de las tramas, dado que este algoritmo, junto a su probabilidad de errores, pudo encontrar uno o más errores dentro de los strings.
2. Es importante determinar cuando usar un algoritmo de detección de errores y cuando usar un algoritmo de corrección de errores, dado que en alguno casos se puede trabajar con canales/medios inseguros, mientras que en otras ocasiones se puede trabajar con medios/canales que son privados y altamente confiables por los emisores y receptores.
3. La probabilidad del ruido afecta mucho en la detección de los errores y la corrección de los mismos, dado que la probabilidad de error en base a la longitud de la cadena era el principal agente que afectaba propiamente a las tramas enviadas y recibidas.

Bibliografía

GeeksforGeeks. (s.f.). Error Detection in Computer Science. Recuperado de: https://www.geeksforgeeks.org/error-detection-in-computer-networks/

Humphrys, M. (s.f.) Error detection and correction methods only work below a certain error rate. Recuperado de: https://humphryscomputing.com/Notes/Networks/data.error.html