Renatto Esteban Guzmán Sosa - 21646
Herber Sebastián Silva Muñoz - 21764
Redes

Laboratorio 2

Escenario de Pruebas 1:
Enviar un mensaje al emisor, copiar el mensaje generado por este y proporcionar tal cual al receptor, el cual debe mostrar el mensajes originales(ya que ningún bit sufrió un cambio). Realizar esto para tres mensajes distintos con distinta longitud.

```
----------------------ESCENARIO 1----------------------
Escenario 1 (Hamming): 10011
Emisor: 101100111
Receptor: No errors detected. Original message: 10011


Escenario 1 (Hamming): 1011011010001
Emisor: 001101110110100101
Receptor: No errors detected. Original message: 1011011010001


Escenario 1 (Hamming): 1101
Emisor: 1010101
Receptor: No errors detected. Original message: 1101


Escenario 1 (CRC32): 10011
Emisor: 10011010101101011110010101110010100 11
Receptor: No errors detected. Original message: 10011


Escenario 1 (CRC32): 1011011010001
Emisor: 10110110100011001100111111011000111011001 10011
Receptor: No errors detected. Original message: 1011011010001


Escenario 1 (CRC32): 1101
Emisor: 11011010110010110011100100110011 0000
Receptor: No errors detected. Original message: 1101
```

Como se puede observar, en este caso, el mensaje que devuelve el receptor es idéntico al que se envió al emisor para que enviase, por lo que la implementación tanto de emisor como de receptor son correctas, donde estos fueron los tres inputs distintos:

```
input1="10011"
input2="1011011010001"
input3="1101"
```

Y aquí se muestra el paso de mensajes manuales entre los distintos lenguajes, usando la salida y entrada estándar de la consola:

```
echo -e "${BOLD_WHITE}Escenario 1 (Hamming): $input1${RESET}"
emitter1=$(python3 emitter.py -Ha $input1)
echo -e "${BOLD_MINT_GREEN}Emisor: $emitter1${RESET}"
receiver1=$(./receiver $emitter1 -Ha)
echo -e "${BOLD_CYAN}Receptor: $receiver1${RESET}"
```

input es el mensaje que se le envía al emisor, emitter es el código concatenado que envía el emisor, y receiver es el mensaje decodificado que interpreta el receptor.

Escenario de pruebas 2:
Enviar un mensaje al emisor, copiar el mensaje generado por este y cambiar un bit cualquiera antes de proporcionarlo al receptor. Si el algoritmo es de detección debe mostrar que se detectó un error y que se descarta el mensaje. Si el algoritmo es de corrección debe corregir el bit, indicar su posición y mostrar el mensaje original. Realizar esto para tres mensajes distintos con distinta longitud.

```
silva@Sebastin:~/Lab2-Redes$ ./scene2.sh
----------------------ESCENARIO 2----------------------
Escenario 2 (Hamming): 10011
Emisor: 101100111
Emisor con bit modificado: 101100101
Receptor: Error detected at position: 8
Error corrected at position: 8
Errors detected and corrected. Corrected message: 10011


Escenario 2 (Hamming): 1011011010001
Emisor: 001101110110100101
Emisor con bit modificado: 000101110110100101
Receptor: Error detected at position: 3
Error corrected at position: 3
Multiple errors detected at positions: 1 2
Errors detected and corrected. Corrected message: 1011011010001


Escenario 2 (Hamming): 1101
Emisor: 1010101
Emisor con bit modificado: 0010101
Receptor: Error detected at position: 1
Error corrected at position: 1
Errors detected and corrected. Corrected message: 1101


Escenario 2 (CRC32): 10011
Emisor: 10011010101101011110010101110010100011
Emisor con bit modificado: 10011010101101011110010101110110100011
CRC-32 checksum mismatch. Data may be corrupted. Message discarded.
Receptor:


Escenario 2 (CRC32): 1011011010001
Emisor: 10110110100011001100111110110001110110011000011
Emisor con bit modificado: 10110110100011001100111110110001110110001000011
CRC-32 checksum mismatch. Data may be corrupted. Message discarded.
Receptor:


Escenario 2 (CRC32): 1101
Emisor: 11011010110010110011100100110011000000
Emisor con bit modificado: 11011010111010110011100100110011000000
CRC-32 checksum mismatch. Data may be corrupted. Message discarded.
Receptor:
```

En este caso a todos los mensajes enviados por el emisor se les modificó un solo bit aleatoriamente, de forma que tuviese errores, esto se hizo con una función en bash, que toma el valor del emisor y lo manipula, luego sigue su curso hacia el receptor.

```
flip_random_bit() {
    local binary_string=$1
    local length=${#binary_string}
    local random_index=$((RANDOM % length))
    local flipped_bit

    if [ "${binary_string:$random_index:1}" == "0" ]; then
        flipped_bit="1"
    else
        flipped_bit="0"
    fi

    echo "${binary_string:0:$random_index}${flipped_bit}${binary_string:$(($random_index + 1))}"
}
```

Acá la muestra de la manipulación manual, en la que muestra cómo llega el mensaje modificado al receptor:

```
echo -e "${BOLD_WHITE}----------------------ESCENARIO 2----------------------${RESET}"

echo -e "${BOLD_WHITE}Escenario 2 (Hamming): $input1${RESET}"
emitter1=$(python3 emitter.py -Ha $input1)
echo -e "${BOLD_MINT_GREEN}Emisor: $emitter1${RESET}"
flipped_emitter1=$(flip_random_bit $emitter1)
echo -e "${BOLD_MINT_GREEN}Emisor con bit modificado: $flipped_emitter1${RESET}"
receiver1=$(./receiver $flipped_emitter1 -Ha)
echo -e "${BOLD_CYAN}Receptor: $receiver1${RESET}"

echo -e "\n"
```

Como se puede observar, los algoritmos de corrección corrigen, y los de detección solo muestran que el mensaje tiene un error

```
Escenario 2 (Hamming): 1101
Emisor: 1010101
Emisor con bit modificado: 0010101
Receptor: Error detected at position: 1
Error corrected at position: 1
Errors detected and corrected. Corrected message: 1101


Escenario 2 (CRC32): 10011
Emisor: 10011010101101011110010101110010100 11
Emisor con bit modificado: 10011010101101011110010101110110100 11
CRC-32 checksum mismatch. Data may be corrupted. Message discarded.
Receptor:
```

Escenario de Pruebas 3:
Enviar un mensaje al emisor, copiar el mensaje generado por este y cambiar dos o más bits cualesquiera antes de proporcionarlo al receptor. Si el algoritmo es de detección debe mostrar que se detectó un error y que se descarta el mensaje. Si el algoritmo es de corrección y puede corregir más de un error, debe corregir los bits, indicar su posición y mostrar el mensaje original. Realizar esto para tres mensajes distintos con distinta longitud.

```
Escenario 3 (CRC32): 1011011010001
Emisor: 10110110100011001100111110110001110100110011
Emisor con bit modificado: 01110110101010100000010110100001010111010010
CRC-32 checksum mismatch. Data may be corrupted. Message discarded.
Receptor:


Escenario 3 (CRC32): 1101
Emisor: 11011010110010110011100100110010000
Emisor con bit modificado: 11010110110010110011100100110111000
CRC-32 checksum mismatch. Data may be corrupted. Message discarded.
Receptor:
```

En este caso se utilizó lo mismo para manipular la información, solo que se hizo flip a dos bits o más, pero menos de un tercio de la cadena, acá lo que observamos es que si bien en ocasiones puede corregirse, en otras, la información puede cambiar totalmente de significado debido a los bits que se cambian.

Escenario de Prueba 4:
¿Es posible manipular los bits de tal forma que el algoritmo seleccionado no sea capaz de detectar el error? ¿Por qué sí o por qué no? En caso afirmativo, demuestrelo con su implementación.

Sí es posible, modificar bits, en especial los de paridad, o cambiar bits específicos puede ocasionar, que no solo se haga un error, sino que se cambie el significado de el mensaje de forma correcta, por lo que entre medias podría decirse, que los algoritmos de corrección no serán capaces de corregir el error o incluso detectarlo, pero es probable que los algoritmos de detección, sí los puedan detectar, puesto que generan más información redundante de los datos, acá un ejemplo:

```
silva@Sebastin:~/Lab2-Redes$ ./scene4.sh
----------------------ESCENARIO 4----------------------
Escenario 4 (Hamming): 1001
Emisor: 0011001
Emisor con bits modificados: 0011001
Receptor: No errors detected. Original message: 1001


Escenario 4 (CRC32): 1001
Emisor: 10011010101111011110010101110010101001
Emisor con bits modificados: 10111010101111010111010101110010101101
CRC-32 checksum mismatch. Data may be corrupted. Message discarded.
Receptor:


En este caso, el error fue indetectable por el código de Hamming.
```

En base a las pruebas que realizó, ¿qué ventajas y desventajas posee cada algoritmo con respecto a los otros dos? Tome en cuenta complejidad, velocidad, redundancia (overhead), etc.

Puede decirse que ambos poseen ventajas y desventajas según el punto de vista. Obviamente en estos casos siempre se va a buscar el algoritmo más rápido, simple, robusto y ligero, pero estos son muy complicados de diseñar o implementar, o simplemente no existen. En este caso, el algoritmo de Hamming sirve para detectar y corregir errores, sin embargo, tiene muchas falencias en temas de robustez, puede cambiarse el significado de la información de forma sencilla y de cierta forma hacer un bypass de este. De la misma forma conforme se fue indagando en el laboratorio, se encontró que a cadenas más largas, este comienza a fallar frecuentemente, por lo que no es tan robusto en esos casos. Una ventaja de este es que es sencillo de implementar. Por otro lado, hablando del algoritmo CRC-32, para detectar errores, la desventaja es que no puede corregirlos, pero es de gran ayuda para conservar la integridad de la información, una de las desventajas es que es demasiado redundante y lento, puesto que la longitud de la cadena crece bastante, además que es más complejo de implementar con respecto a los demás, sin embargo, una de sus ventajas es que es robusto.