Universidad del Valle de Guatemala

Facultad de ingeniería

Redes

Catedrático: Jorge Yass

Laboratorio #2 - Primera parte

Esquemas de detección y corrección de errores

Nelson Eduardo García Bravatti 22434

Joaquín André Puente 22296

Guatemala, julio de 2025

Algoritmos Implementados:

Corrección de errores

- Código de Hamming
- Códigos convolucionales (Algoritmo de Viterbi)

Detección de errores

- Fletcher checksum
- CRC-32

Pruebas:

Algoritmo de Hamming

Mensaje	Condición	Codificado	Modificado	Resultado
1101	**Sin errores**	**1100101**	**N/A**	**No se detectaron errores. Mensaje decodificado: 1101**
110110101	**Sin errores**	**11001011111010**	**N/A**	**No se detectaron errores. Mensaje decodificado: 11011010**
101011001110	**Sin errores**	**011101010111010**	**N/A**	**No se detectaron errores. Mensaje decodificado: 101011001110**
1101	**1 error**	**1100101**	**1100111**	**Error detectado en posición 6. Mensaje corregido: 1101**
110110101	**1 error**	**11001011111010**	**11001010111010**	**Error detectado en posición 7. Mensaje**

				corregido: 11011010
101011001110	**1 error**	**011101010111010**	**011101011111010**	**Error detectado en posición 2. Mensaje corregido: 101011001110**
1101	**2 o más errores**	**1100101**	**1110111**	**Error detectado en posición 4. Mensaje INCORRECTO: 1111**
110110101	**2 o más errores**	**11001011111010**	**11011011101010**	**Error detectado en posición 6. Mensaje INCORRECTO**
101011001110	**2 o más errores**	**011101010111010**	**111101011111010**	**Error detectado en posición 1. Mensaje INCORRECTO**

¿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

Cambiar bits en posiciones que anulan los síndromes (bits 3, 6, 7)

Original: 1100101

Corrupto: 1111111 (cambios en posiciones 3, 6, 7)

El ejemplo anterior es un ejemplo donde se puede ver que el algoritmo no fue capaz de detectar el error.

Ventajas:

- **Corrección automática**: Única ventaja significativa - puede corregir errores de 1 bit sin retransmisión
- **Velocidad de procesamiento**: Operaciones XOR simples, muy rápido para bloques pequeños
- **Latencia baja**: No requiere retransmisión en errores simples
- **Implementación directa**: Lógica de corrección relativamente simple

Desventajas:

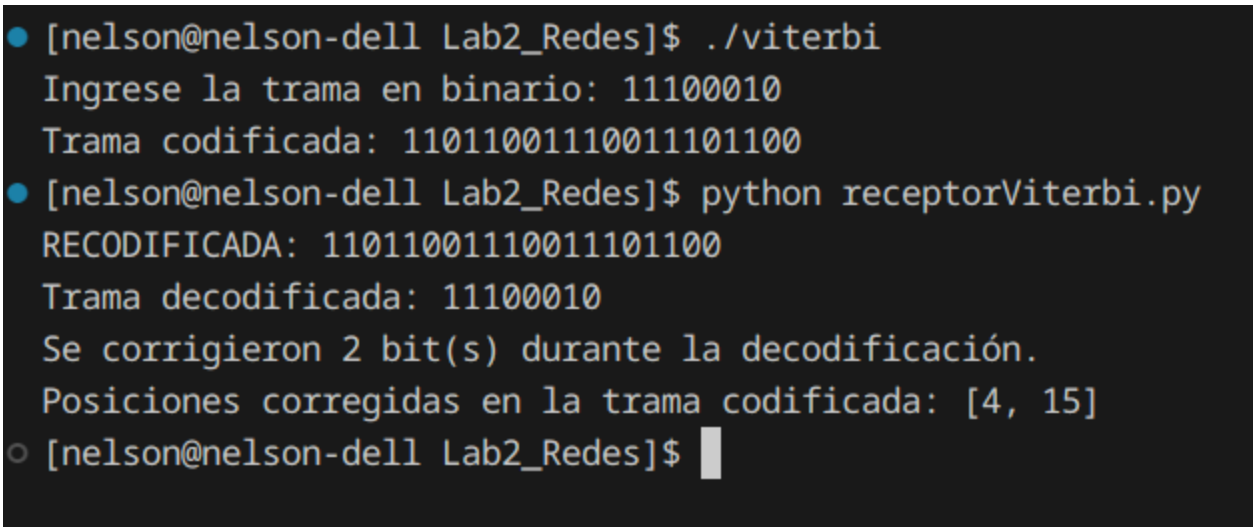
- **Redundancia alta**: 75% de overhead constante (3 bits por cada 4 de datos)
- **Capacidad limitada**: Solo detecta/corrige 1 bit por bloque de 7

- **Vulnerabilidad a errores múltiples**: Fácilmente engañado por patrones específicos
- **Escalabilidad pobre**: Overhead no mejora con mensajes más largos
- **Confiabilidad cuestionable**: Durante las pruebas, errores de 2+ bits produjeron resultados incorrectos sin advertencia

Algoritmo de Viterbi:

Mensaje	Condición	Codificado	Modificado	Resultado
1010	Sin errores	Trama codificada: 111000101100	N/A	Trama decodificada: 1010 No se detectaron errores.
1111111	Sin errores	Trama codificada: 110110101010100111	N/A	Trama decodificada: 1111111 No se detectaron errores.
10110011	Sin errores	Trama codificada: 111000010111110101111	N/A	Trama decodificada: 10110011 No se detectaron errores.
1111	1 error	Trama codificada: 110110100111	110110100110	Trama decodificada: 1111 Se corrigieron 1 bit(s) durante la decodificación. Posiciones corregidas en la trama codificada: [11]
101010	1 error	Trama codificada: 111000100010100	111000101010100	Trama decodificada: 101010 Se corrigieron 1 bit(s) durante la decodificación. Posiciones corregidas en la trama codificada: [8]

110001110	1 error	Trama codificada: 1101011100110 110011100	1111011100110 110011100	Trama decodificada: 110001110 Se corrigieron 1 bit(s) durante la decodificación. Posiciones corregidas en la trama codificada: [2]
10001	2 o más errores	Trama codificada: 1110110011101 1	1100110011111 1	Trama decodificada: 10001 Se corrigieron 2 bit(s) durante la decodificación. Posiciones corregidas en la trama codificada: [2, 11]
11100010	2 o más errores	Trama codificada: 1101100111001 1101100	1101100111001 1101111	Trama decodificada: 11100010 Se corrigieron 2 bit(s) durante la decodificación. Posiciones corregidas en la trama codificada: [18, 19]
10101000	2 o más errores	Trama codificada: 1110001000101 1000000	1010001000101 1010000	Trama decodificada: 10101000 Se corrigieron 2 bit(s) durante la decodificación. Posiciones corregidas en la trama codificada: [1, 15]



Modificada: 11010001110011111100

¿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.

R//

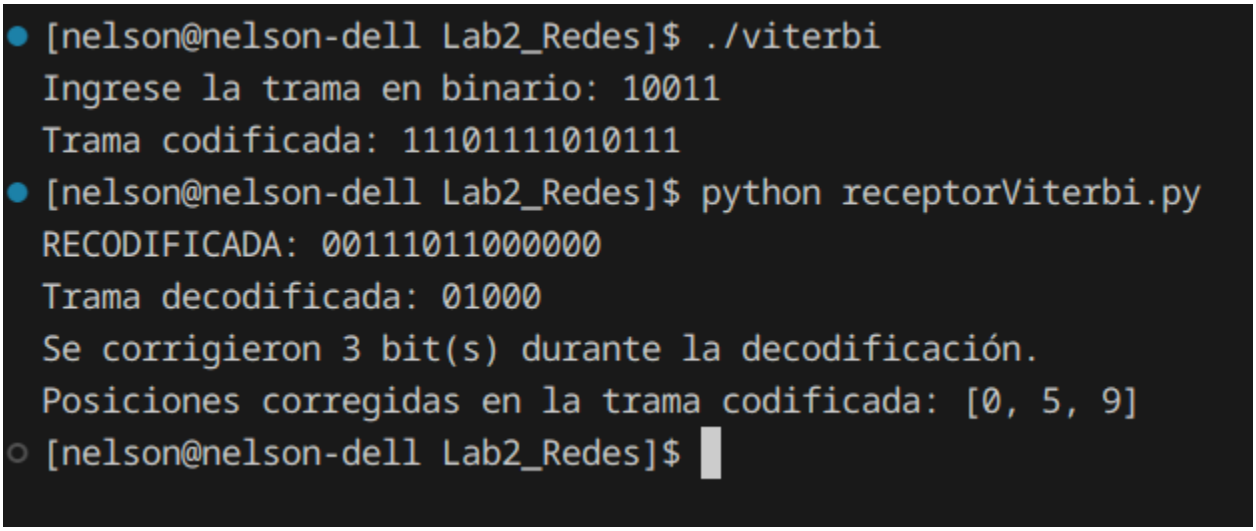
El algoritmo de Viterbi se utiliza para decodificar códigos convolucionales y está diseñado para encontrar la secuencia de bits más probable que se haya transmitido, basándose en la métrica de distancia (generalmente la distancia de Hamming en decodificación dura). La capacidad del algoritmo para detectar y corregir errores depende de la distancia libre d_free del código convolucional, que es la mínima distancia de Hamming entre dos secuencias válidas generadas por el código.

Como se implementó en este laboratorio, el código convolucional tiene K=3 (longitud de restricción), con polinomios generadores G1=111 y G2=101. La distancia libre de este código es d_free = 9, lo que significa que el código puede corregir hasta |(d_free - 1)/2| (piso) = |(9-1)/2| (piso) = 4 errores por bloque en decodificación dura, asumiendo que los errores están suficientemente separados.

Sí, es posible. Esto ocurre si se introduce un número de errores mayor al límite de corrección del código (en este caso 4) o si los errores están distribuidos de tal manera que la secuencia recibida es más cercana (en términos de distancia de Hamming) a una secuencia válida diferente de la original. En este caso, el algoritmo de Viterbi seleccionará la secuencia válida más cercana, lo que resultará en una decodificación incorrecta sin detectar que hubo un error, ya que la métrica de distancia será mínima para esa secuencia incorrecta.

Ejemplo:

Se modifican más de 4 bits: 10111111010000



En este caso los resultados muestran otra trama que no corresponde a la original.

Fletcher Checksum:

Mensaje	Condición	Codificado	Modificado	Resultado
1010	Sin errores	Mensaje + checksum: 1010C2E7	N/A	No se detectaron errores. Trama recibida: 1010
1111111	Sin errores	Mensaje + checksum: 11111115861	N/A	No se detectaron errores. Trama recibida: 1111111
10110011	Sin errores	Mensaje + checksum: 1011001186DC	N/A	No se detectaron errores. Trama recibida: 10110011
1111	1 error	Mensaje + checksum: 1111C4EB	1101C4EB	Se detectó error. Trama descartada.
101010	1 error	Mensaje + checksum: 1010102400	1110102400	Se detectó error. Trama descartada.
110001110	1 error	Mensaje + checksum: 110001110B692	110101110B692	Se detectó error. Trama descartada.

10001	2 o más errores	Mensaje + checksum: 10001F2D8	10111F2D8	Se detectó error. Trama descartada.
11100010	2 o más errores	Mensaje + checksum: 1110001085DD	1100000085DD	Se detectó error. Trama descartada.
10101000	2 o más errores	Mensaje + checksum: 1010100084D8	1110110084D8	Se detectó error. Trama descartada.

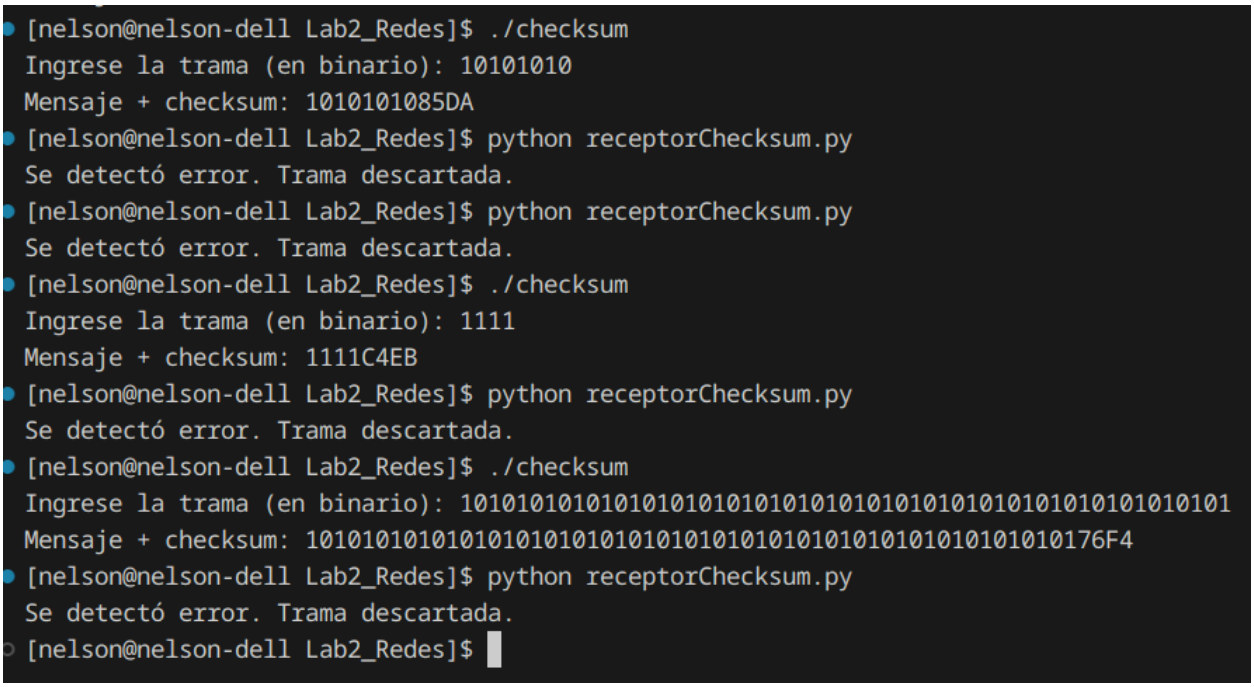
```
[nelson@nelson-dell Lab2_Redes]$ ./checksum
Ingrese la trama (en binario): 10001
Mensaje + checksum: 10001F2D8
[nelson@nelson-dell Lab2_Redes]$ python receptorChecksum.py
Se detectó error. Trama descartada.
```

¿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.

R//

El Fletcher Checksum es un algoritmo lineal. Esto significa que, si haces cambios compensatorios en el mensaje, puedes mantener el mismo resultado de checksum. Por ejemplo, si incrementas un byte del mensaje y decrementas otro, en muchos casos el resultado del checksum no cambia. En general, los checksums como el de Fletcher no son criptográficamente seguros, y solo protegen contra errores aleatorios simples (como 1 bit cambiado, o un solo carácter modificado), pero pueden ser vulnerados ante ataques intencionales o errores múltiples.

Aunque en teoría el algoritmo Fletcher-16 puede ser engañado con manipulaciones específicas de los datos, en la práctica (para esta implementación, con tramas cortas y usando solo caracteres '0' y '1' en ASCII), es muy poco probable encontrar colisiones con solo uno o dos cambios. Para encontrar colisiones prácticas se requerirían mensajes más largos y cambios planeados en varios bits, o manipular bytes arbitrarios (no solo '0' y '1').



Algoritmo CRC32

Mensaje	Condición	Codificado	Modificado	Resultado
1101	Sin errores	1101111000101 1000000100011 1011011011	N/A	No se detectaron errores. Mensaje extraído: 1101
110110101	Sin errores	1101101010000 0100110000010 0011101101101 11	N/A	No se detectaron errores. Mensaje extraído: 110110101
101011001110	Sin errores	1010110011100 0001000110000 0100011101101 10111	N/A	No se detectaron errores. Mensaje extraído: 101011001110
1101	1 error	1101111000101 1000000100011 1011011011	1101111000101 1000000100011 1011011010	Se detectó error. Mensaje descartado
110110101	1 error	1101101010000 0100110000010 0011101101101 11	1101101010000 0100110000010 0011101101101 10	Se detectó error. Mensaje descartado
101011001110	1 error	1010110011100 0001000110000 0100011101101 10111	1010110011100 0001000110000 0100011101101 10110	Se detectó error. Mensaje descartado

1101	2 o más errores	1101111000101 1000000100011 1011011011	1100111000101 1000000100011 1011011010	Se detectó error. Mensaje descartado
110110101	2 o más errores	1101101010000 0100110000010 0011101101101 11	1100101011000 0100110000010 0011101101101 10	Se detectó error. Mensaje descartado
101011001110	2 o más errores	1010110011100 0001000110000 0100011101101 10111	1011111011100 0001000110000 0100011101101 10110	Se detectó error. Mensaje descartado

¿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

CRC-32 no puede detectar errores cuando el patrón de error es exactamente un múltiplo del polinomio generador. Si los errores forman un patrón que, al dividirse por el polinomio CRC-32, da resto cero, entonces el error no será detectado.

Ventajas

- **Overhead escalable**: Muy eficiente para mensajes largos (solo 3.2% para 1000 bits vs 75% de Hamming)
- **Detección robusta**: En las pruebas, detectó TODOS los errores aleatorios probados (1, 2, 3, 4+ bits)
- **Confiabilidad matemática**: Probabilidad de error no detectado de 2^-32
- **Estándar industrial**: Probado en millones de implementaciones (Ethernet, TCP/IP, ZIP)
- **Detección de ráfagas**: Puede detectar errores consecutivos hasta 32 bits
- **Simplicidad de decisión**: Solo dos estados: válido o inválido

Desventajas:

- **Solo detección**: No puede corregir errores automáticamente
- **Overhead inicial alto**: Para datos muy pequeños (<32 bits), el overhead puede ser >100%
- **Complejidad computacional**: División polinomial más compleja que XOR simple
- **Requiere retransmisión**: En caso de error, necesita mecanismo adicional de recuperación
- **Latencia en errores**: Detección de error implica retransmisión completa