

**UNIVERSIDAD DEL VALLE DE GUATEMALA**

**Facultad de Ingeniería**

**Redes – Jorge Yass**



*Excelencia que trasciende*

**DEL VALLE**  
GRUPO EDUCATIVO

**Laboratorio 2**

Daniel Alfredo Rayo Roldán, 22933

Irving Fabricio Morales Acosta, 22781

## Github

<https://github.com/DanielRasho/redes-lab2>

## Pruebas

Durante este laboratorio se trabajó en la práctica *Esquemas de detección y corrección de errores* implementando y comparando dos esquemas de detección de errores, CRC-32, y otro de corrección de errores, Viterbi) bajo una arquitectura de capas. Se desarrolló un Emisor en Python con capas de Aplicación (ingreso de texto y selección de algoritmo), Presentación (conversión ASCII - binario), Enlace (cálculo y verificación de CRC-32 ó codificación convolucional) y Ruido (inversión de bits según probabilidad). En paralelo, se construyó un Receptor en Go, siempre escuchando en un puerto TCP para decodificar la trama recibida y notificar aciertos o errores. Finalmente, se diseñaron pruebas automáticas que envían mensajes de distintas longitudes, aplican ruido con diversas tasas de error y registran la tasa de éxito, el overhead y el tiempo de respuesta.

A continuación, se muestran los resultados de las pruebas realizados a ambos esquemas.

### CRC-32

Al aplicar CRC-32 sobre mensajes de 16 bytes la tasa de éxito cae rápidamente incluso con un 1 % de error. En ausencia de ruido (0 %), el receptor descifra correctamente el 86 % de los mensajes, pero con solo un 1 % de inversión de bits ya baja al 11 % y a 0 % a partir del 2 %.

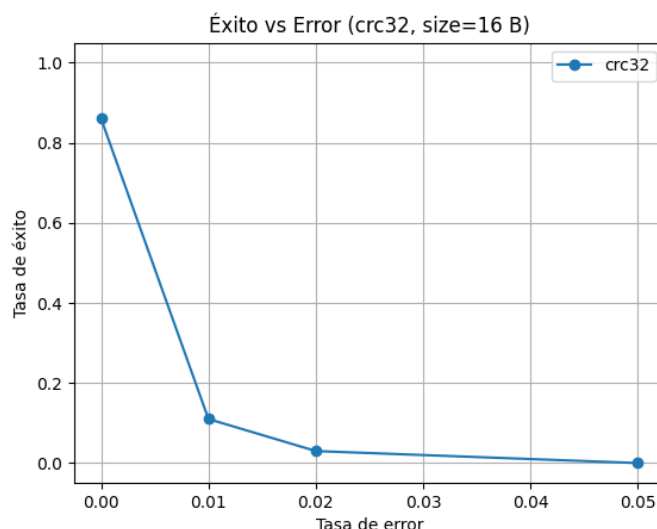


Figura 1. Éxito vs Error (crc32, tamaño = 16 B).

Para 128 B sin ruido se alcanza apenas 24 % de éxito, y con cualquier tasa  $\geq 1\%$  la detección falla en el 100 % de los casos.

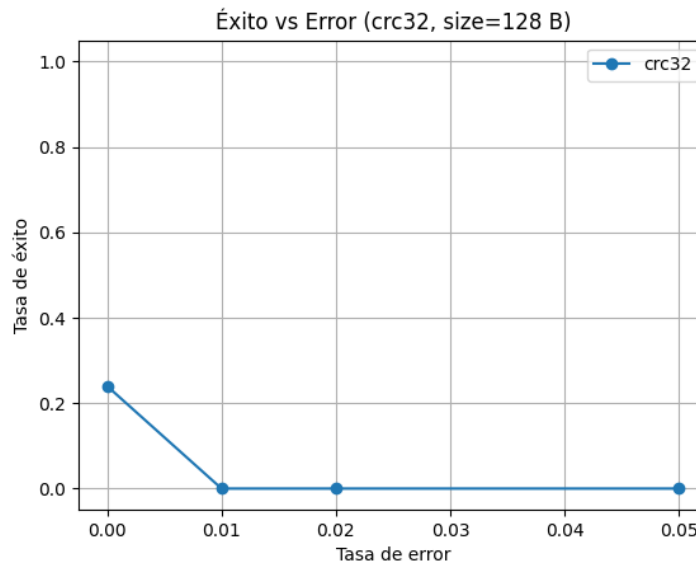


Figura 2. Éxito vs Error (crc32, tamaño = 128 B).

Con 256 bytes la tasa inicial baja al 11 % (0 % de ruido) y desaparece completamente al introducir un 1 % de errores.

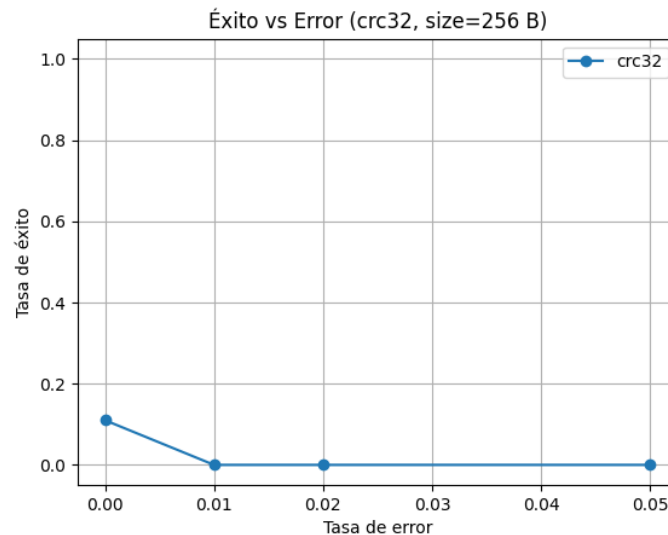


Figura 3. Éxito vs Error (crc32, tamaño = 256 B).

Este esquema de detección resulta muy sensible a partir de tasas de error bajas ( $< 1\%$ ), especialmente para mensajes de tamaño medio o grande, solo en mensajes muy cortos y

sin ruido muestra éxito moderado. Aunque eso dice más de la calidad del medio que del algoritmo utilizado.

## Viterbi

El decodificador Viterbi logra corregir hasta tasas alrededor de 2 %. SU tasa de éxito comienza en 85%, sube ligeramente a 87 % con 1 % de error y se mantiene en 80 % con 2%. Solo al 5 % cae al 55 %.

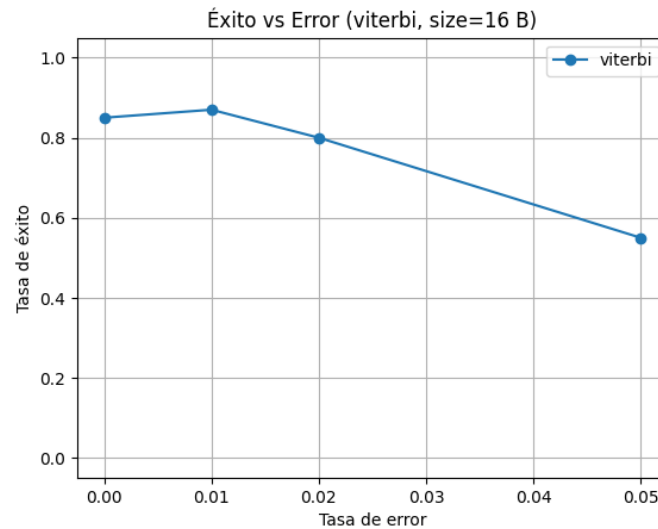


Figura 4. Éxito vs Error (viterbi, tamaño = 16 B).

Con bloques de 128 bytes, la efectividad inicial es menor (19 % sin ruido), mejora a 29 % con 1 % de error y vuelve a 19 % al 2 %, al 5 % prácticamente no corrige.

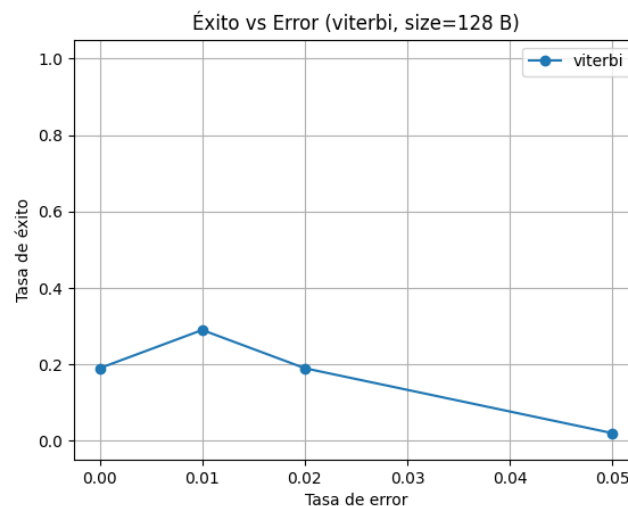


Figura 5. Éxito vs Error (viterbi, tamaño = 128 B).

Para 256 bytes la corrección empieza en 13 % de éxito (0 % de ruido), baja a 6 % al 1 %, 5% al 2 % y se anula al 5 %.

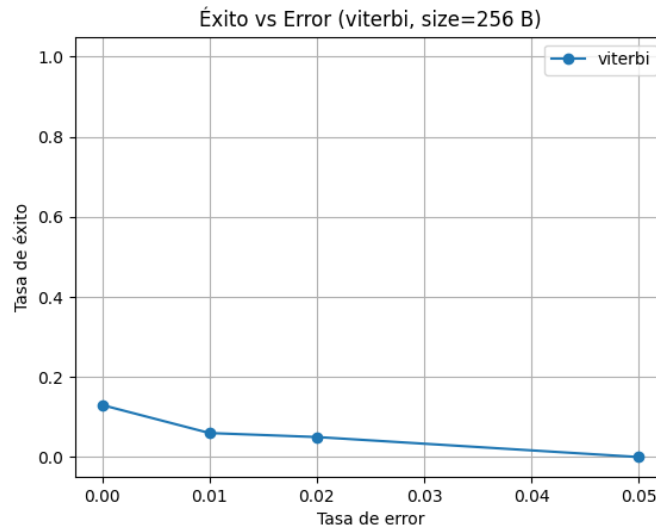


Figura 6. Éxito vs Error (viterbi, tamaño = 256 B).

Viterbi corrige errores con tasas moderadas (1–2 %) en mensajes muy cortos, su rendimiento base (incluso sin ruido) es bajo para bloques de mayor tamaño debido al overhead y la complejidad de decodificación.

### Discusión

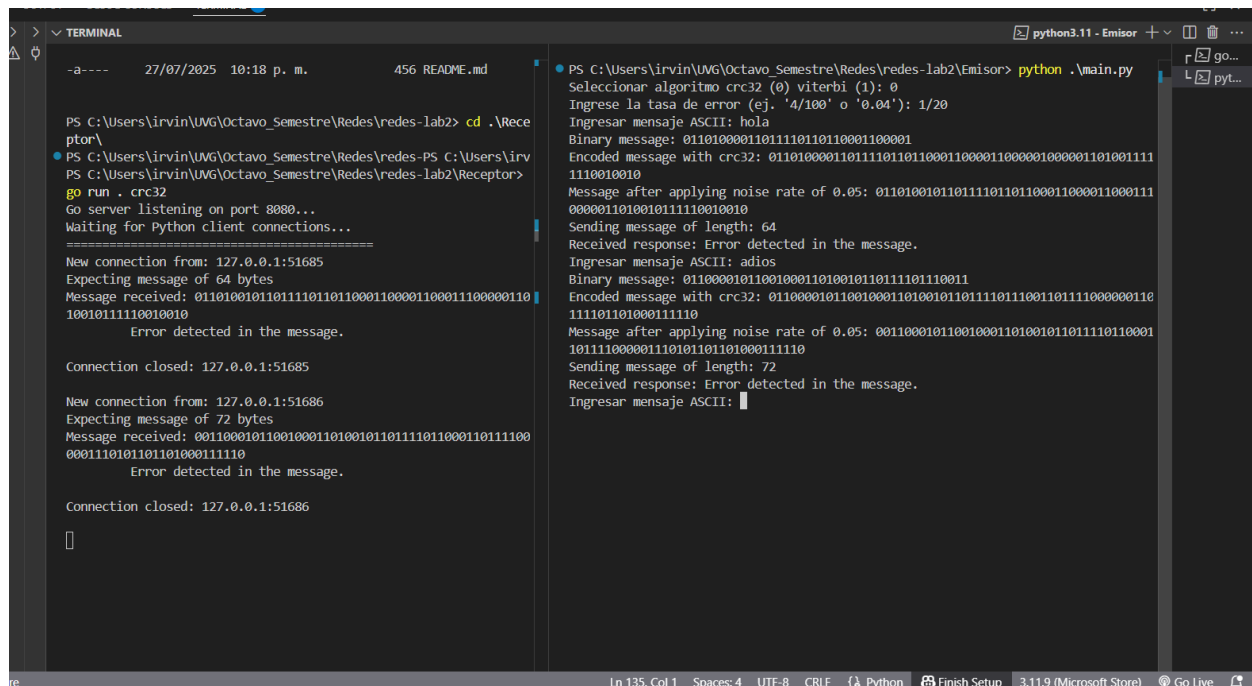
Tras observar los resultados de ambos esquemas se puede observar cómo ninguna es mejor que la otra. A pesar de que CRC-32 es simple y rápido (debido a su bajo coste computacional), no es capaz de corregir los errores, solo los detecta. En las figuras 1-3 se observa como a partir desde un 1% de ruido la tasa de éxito decaiga a casi 0, incluso con mensajes cortos de 16 bytes.

Por otro lado, el esquema Viterbi introduce un overhead importante que llega a ser el doble del número de bytes originales y su coste computacional no es despreciable, lo que se ve reflejado en la tasa de éxito “base”, es decir, cuando no hay ruido ya que bloques grandes tienen una baja probabilidad de reconstrucción gracias la longitud de la secuencia codificada. Sin embargo, Viterbi sí corrige los errores de canal y logra un éxito entre 80%-90% aproximadamente 80%-90% en tramas cortas y con un ruido moderado de aproximadamente 1%-2%.

## Conclusiones

- Para canales casi libres de errores o tramas muy cortas, CRC-32 es suficiente es una buena elección por su eficiencia.
- En canales con ruido moderado ( $< 2\%$ ) y mensajes críticos (aunque muy breves), Viterbi ofrece corrección sin retransmisiones, por lo que hace buena opción.
- En escenarios de mensajes grandes o ruido elevado ( $> 5\%$ ), ninguna de los dos esquemas son fiables por sí sola.
- En la práctica, lo ideal es combinar ambas aproximaciones o recurrir a esquemas híbridos.

## Anexos



```
-a---- 27/07/2025 10:18 p. m. 456 README.md

PS C:\Users\irvin\UWG\Octavo_Semestre\Redes\redes-lab2> cd .\Receptor\
PS C:\Users\irvin\UWG\Octavo_Semestre\Redes\redes-lab2\Receptor> go run .\crc32
Go server listening on port 8080...
Waiting for Python client connections...
=====
New connection from: 127.0.0.1:51685
Expecting message of 64 bytes
Message received: 01101001011011101100011000011000011000000110
10010111110010010
Error detected in the message.

Connection closed: 127.0.0.1:51685

New connection from: 127.0.0.1:51686
Expecting message of 72 bytes
Message received: 001100010110010001101001011011101100011011100
00011010110110000111110
Error detected in the message.

Connection closed: 127.0.0.1:51686

PS C:\Users\irvin\UWG\Octavo_Semestre\Redes\redes-lab2\Emisor> python .\main.py
Seleccionar algoritmo crc32 (0) viterbi (1): 0
Ingresar la tasa de error (ej. '4/100' o '0.04'): 1/20
Ingresar mensaje ASCII: hola
Binary message: 01101000011011110110110001100001
Encoded message with crc32: 011010000110111101101100011000011000001101001111
1110010010
Message after applying noise rate of 0.05: 01101001011011110110001100001100011
000001101001011110010010
Sending message of length: 64
Received response: Error detected in the message.
Ingresar mensaje ASCII: adios
Binary message: 011000010110010001101001011011101110011
Encoded message with crc32: 0110000101100100011010010110111011100110111000000110
111101101000111110
Message after applying noise rate of 0.05: 00110001011001000110100101101110110001
1011110000011101011011000111110
Sending message of length: 72
Received response: Error detected in the message.
Ingresar mensaje ASCII: 
```

Programa corriendo con CRC32 del lado receptor y CRC32 del lado emisor

```
19 def apply_algorithm(bit_str: str, algorithm: str = "") -> str:
    """
    Apply a bit manipulation algorithm to a binary string.
    """
    if algorithm == "viterbi":
        return viterbi_decode(bit_str)
    elif algorithm == "crc32":
        return crc32_decode(bit_str)
    else:
        return bit_str

# Main function
def main():
    # Select algorithm
    algorithm = input("Seleccione algoritmo (0) viterbi (1): ")
    algorithm = int(algorithm)

    # Enter error rate
    error_rate = input("Ingrese la tasa de error (ej. '4/100' o '0.04'): ")
    error_rate = float(error_rate)

    # Enter message
    message = input("Ingrese mensaje ASCII: ")

    # Encode message
    encoded_message = encode_message(message, algorithm)

    # Apply noise
    message_after_noise = apply_noise(encoded_message, error_rate)

    # Decode message
    decoded_message = decode_message(message_after_noise, algorithm)

    # Print result
    print("Mensaje decodificado: ", decoded_message)

# Run the program
if __name__ == "__main__":
    main()
```

PS C:\Users\irvin\UWG\Octavo\_Semestre\Redes\redes-lab2> cd .\Receptor\

PS C:\Users\irvin\UWG\Octavo\_Semestre\Redes\redes-lab2\Receptor> go run .\crc32

Go server listening on port 8080...

Waiting for Python client connections...

=====

New connection from: 127.0.0.1:51685

Expecting message of 64 bytes

Message received: 01101000110111101101100011000011000111000001101001011110010010

Error detected in the message.

Connection closed: 127.0.0.1:51685

New connection from: 127.0.0.1:51686

Expecting message of 72 bytes

Message received: 0011000101100100011010010110111011000110111000011001101101101100011110

Error detected in the message.

Connection closed: 127.0.0.1:51686

New connection from: 127.0.0.1:51792

Expecting message of 64 bytes

Message received: 001101010010110000110101000110101100010100010110001011

Error detected in the message.

Connection closed: 127.0.0.1:51792

PS C:\Users\irvin\UWG\Octavo\_Semestre\Redes\redes-lab2\Emissor> python .\main.py

Seleccione algoritmo crc32 (0) viterbi (1): 1

Ingrese la tasa de error (ej. '4/100' o '0.04'): 2/100

Ingresar mensaje ASCII: hola

Binary message: 01101000011011110110110001100001

Encoded message with viterbi: 001101010010110000110101000110100010100010111001101011000011

Message after applying noise rate of 0.02: 001101010010110000110101000110101100010111000010

1000101110010010111000001

Sending message of length: 64

Received response: Error detected in the message.

Ingresar mensaje ASCII: adios

Binary message: 011000010110010001101001011011110110011

Encoded message with viterbi: 001101011100001110000101111101100110101001011111000

0101000110100100011001111101

Message after applying noise rate of 0.02: 0011010111100111000010110110110011010

10010111110000101000110100100011101101101

Sending message of length: 80

Received response: Error detected in the message.

Ingresar mensaje ASCII:

Programa corriendo con CRC32 del lado receptor y viterbi del lado emisor

```
PS C:\Users\irvin\UWG\Octavo_Semestre\Redes\redes-lab2\Receptor> go run .\viterbi
```

Go server listening on port 8080...

Waiting for Python client connections...

=====

New connection from: 127.0.0.1:52011

Expecting message of 64 bytes

Message received: 011010000110011101101100011000001000001000001101001011110010010

1001111110010010

Received 64 bits.

Decoded sequence: 0101110010000000000001100111010

sequence corrupted: detected 9 bit errors at positions [ 1 3 16 25 26 32 38 49 54]

Corrected bits: 0101110010000000000001100111010

Error positions: [ 1 3 16 25 26 32 38 49 54]

Connection closed: 127.0.0.1:52011

New connection from: 127.0.0.1:52012

Expecting message of 72 bytes

Message received: 011000010110010001101001011011101110011011110011011100

00001101110110100011110

Received 72 bits.

Decoded sequence: 00011011110100110110000010011110010

sequence corrupted: detected 9 bit errors at positions [ 1 2 6 25 34 38 44 50 65]

Corrected bits: 00011011110100110110000010011110010

Error positions: [ 1 2 6 25 34 38 44 50 65]

Connection closed: 127.0.0.1:52012

PS C:\Users\irvin\UWG\Octavo\_Semestre\Redes\redes-lab2\Emissor> python .\main.py

Seleccione algoritmo crc32 (0) viterbi (1): 0

Ingrese la tasa de error (ej. '4/100' o '0.04'): 2/100

Ingresar mensaje ASCII: hola

Binary message: 01101000011011110110110001100001

Encoded message with crc32: 011010000110111101101100011000011000001000001101001111

1110010010

Message after applying noise rate of 0.02: 011010000110011101101100011000001000001

000001101001111110010010

Sending message of length: 64

Received response: sequence corrupted: detected 9 bit errors at positions [ 1 3 16 25 26 32 38 49 54]; corrected to bits 0101110010000000000001100111010; error positions [ 1 3 16 25 26 32 38 49 54]

Ingresar mensaje ASCII: adios

Binary message: 011000010110010001101001011011110110011

Encoded message with crc32: 011000010110010001101001011110111001101111000000110

11101101000111110

Message after applying noise rate of 0.02: 011000010110010001101001011011110111001

10111100000011011110110100011110

Sending message of length: 72

Received response: sequence corrupted: detected 9 bit errors at positions [ 1 2 6 25 34 38 44 50 65]; corrected to bits 00011011110100110110000010011110010; error positions [ 1 2 6 25 34 38 44 50 65]

Ingresar mensaje ASCII:

Programa corriendo con Viterbi del lado receptor y CRC32 del lado emisor

```
Decoded sequence: 00011101111010011011000010011110010
sequence corrupted: detected 9 bit errors at positions [1 2 6 25
34 38 44 50 65]
Corrected bits: 00011101111010011011000010011110010
Error positions: [1 2 6 25 34 38 44 50 65]

Connection closed: 127.0.0.1:52012

New connection from: 127.0.0.1:52248
Expecting message of 64 bytes
Message received: 00110101001011000011010100011010010001010001011
10011010111000001
Received 64 bits.
Decoded sequence: 0110100001101111011001100000
sequence corrupted: detected 1 bit errors at positions [63]
Corrected bits: 0110100001101111011001100000
Decoded text: "hol"
Error positions: [63]

Connection closed: 127.0.0.1:52248

New connection from: 127.0.0.1:52250
Expecting message of 80 bytes
Message received: 0011010111000011100001011111011001101010010011
11000010100011010010001100111101
Received 80 bits.
Decoded sequence: 0110000101100100011010010110111101110011
sequence corrupted: detected 1 bit errors at positions [44]
Corrected bits: 0110000101100100011010010110111101110011
Decoded text: "adios"
Error positions: [44]

Connection closed: 127.0.0.1:52250

PS C:\Users\irvin\OneDrive\Octavo_Semestre\Redes\lab2\Emisor> python .\main.py
o Seleccionar algoritmo crc32 (0) viterbi (1): 1
Ingresar la tasa de error (ej. '4/100' o '0.04'): 2/100
Ingresar mensaje ASCII: hola
Binary message: 01101000011011110110110001100001
Encoded message with viterbi: 0011010100101100001101010001101000101000101110011
010111000011
Message after applying noise rate of 0.02: 001101010010110000110101000110100100010
1000101110011010111000001
Sending message of length: 64
Received response: sequence corrupted: detected 1 bit errors at positions [63]; co
rrected to bits 01101000011011110110110001100000 -> "hol"; error positions [63]
Ingresar mensaje ASCII: adios
Binary message: 0110000101100100011010010110111101110011
Encoded message with viterbi: 0011010111000011100001011111011001101010011111000
0101000110100100011001111101
Message after applying noise rate of 0.02: 00110101110000111000010111110110011010
10010011110000101000110100100011001111101
Sending message of length: 80
Received response: sequence corrupted: detected 1 bit errors at positions [44]; co
rrected to bits 0110000101100100011010010110111101110011 -> "adios"; error positio
ns [44]
Ingresar mensaje ASCII: 
```

Programa corriendo con Viterbi del lado receptor y Viterbi del lado emisor

Referencias

Kelche, K. (2023, febrero 14). *A complete guide to socket programming in Go*. Kelche.Co. <https://www.kelche.co/blog/go/socket-programming/>

*socket* — *Low-level networking interface*. (n.d.). Python Documentation. From <https://docs.python.org/3/library/socket.html>

*UConn HKN*— (2017, diciembre 03.). Digital Communications: Viterbi Algorithm . From <https://www.youtube.com/watch?v=dKIf6mQUfnY>