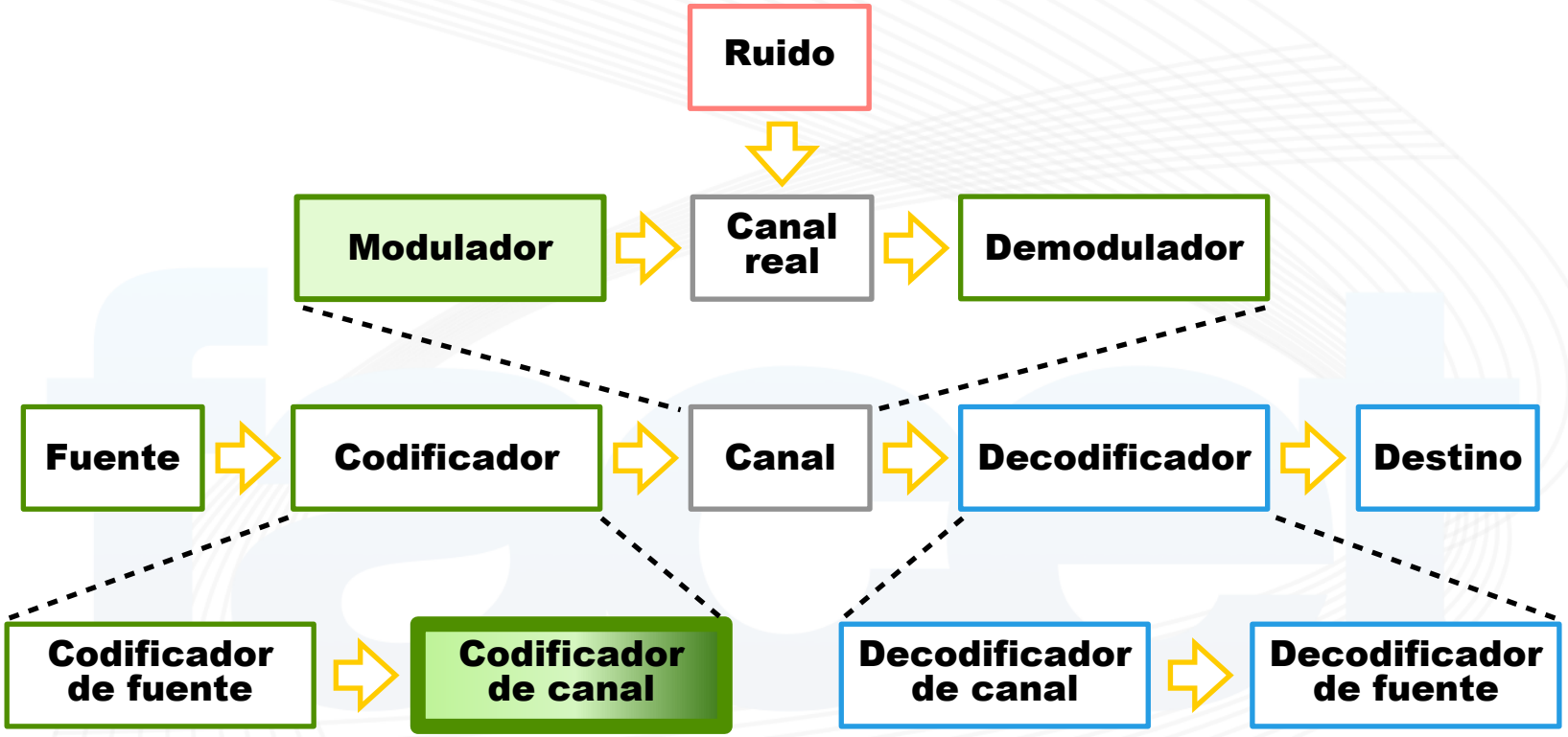
Detección y Corrección de Errores

Transmisión de Datos

Ing. Luis DI PINTO (ldipinto@herrera.unt.edu.ar)

http://www.microprocesadores.unt.edu.ar/transmision/

Repaso Esquema general



- Asumiendo que ya tenemos una señal digital codificada lista para transmitir, la prepararemos mejor previa a su modulación.

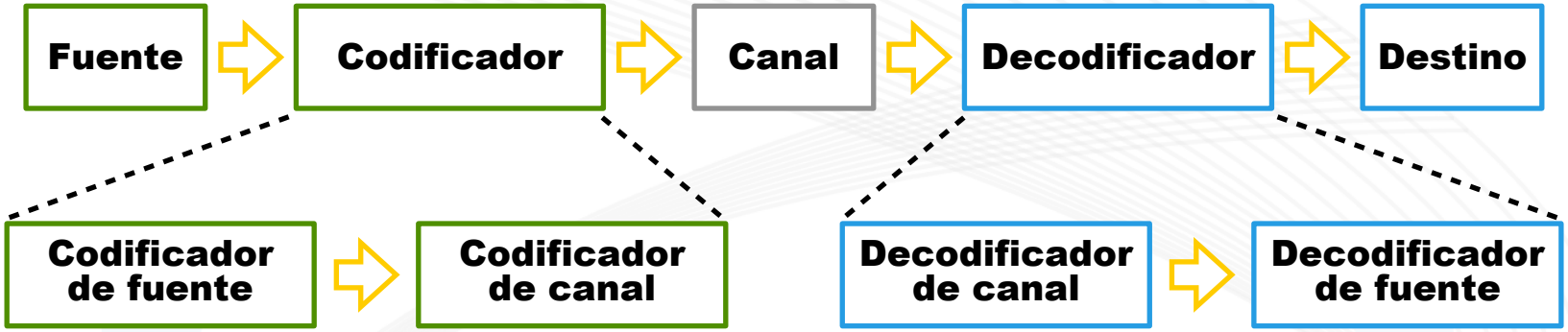
Codificación del Canal

- Para un mejor abordaje, la dividiremos en tres partes:
  - Codificaciones de línea
    - Características vinculadas a capa física y modulación en banda base.
  - Características vinculadas a las modulaciones de portadora.
  - Características vinculadas a la generación de tramas.
    - **Detección y corrección de errores.**

Temas que veremos

- Conceptos y definiciones
- Detección de errores
  - Paridad.
  - CRC.
- Corrección de errores
  - Códigos de Hamming.
  - Códigos convolucionales.

Codificación de fuente y de canal



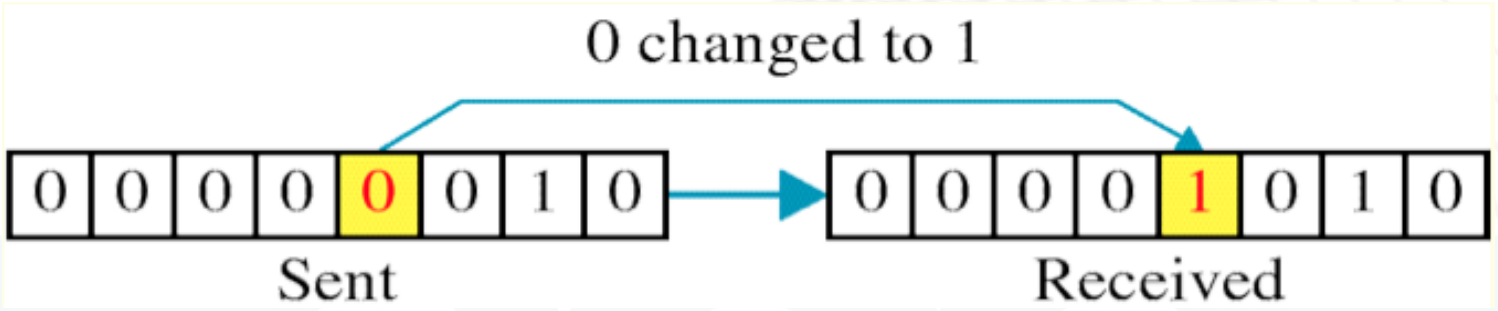
- La decodificación de canal debe **revertir los cambios introducidos por el canal** al transmitir el mensaje.
- Para ello, en la codificación **se agrega información redundante** que le permita detectar y/o corregir los errores en base a estadística.

Tipos de errores

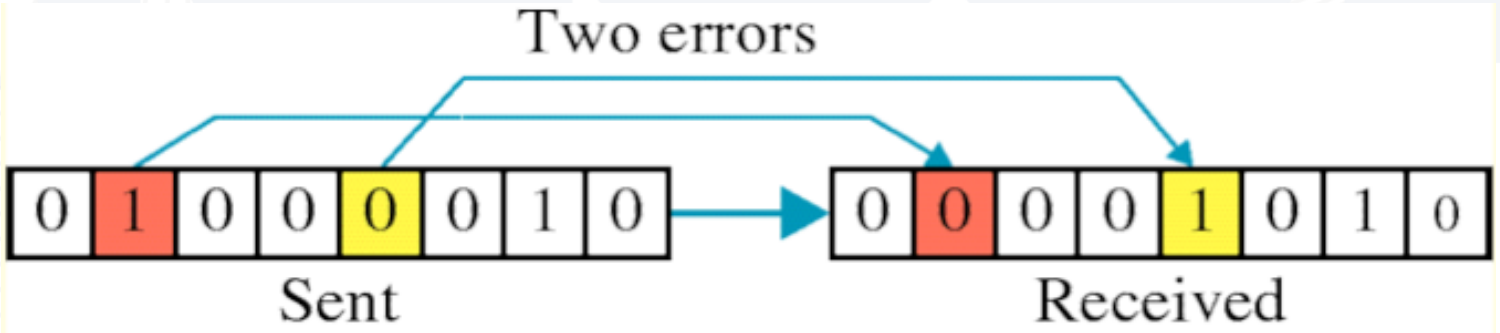
- Los errores se clasifican de la siguiente forma:
  - **Error a nivel de bit:** afecta individualmente a n bits del dato transmitido.
    - En función de la cantidad se puede hablar de error simple, doble, triple, etc.
  - **Errores en ráfaga:** afecta en conjunto a m bits consecutivos del dato transmitido.
    - El primero y el ultimo bit de la ráfaga siempre cambian su valor, pero los bits intermedios pueden o no cambiar el valor original.

Errores a nivel de bits

- Error simple

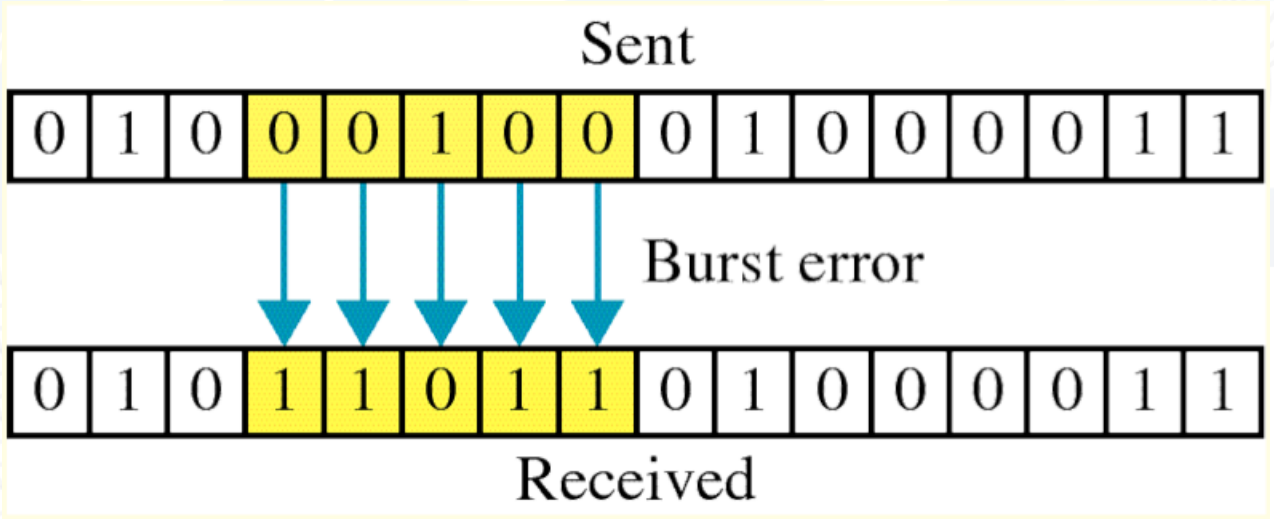


- Error doble



Errores en ráfaga

- El ejemplo supone que todos los bits de la ráfaga son alterados, pero los bits intermedios podrían mantener su valor



Definiciones

- Se denomina **código** a un determinado conjunto de patrones de bits, usualmente de longitud fija.
- Dado dos patrones *p* y *q* que pertenecen a un código, se denomina **distancia de Hamming *d(p,q)*** a la cantidad de posiciones de bits en los que difieren p y q.
- Dado un patrón *p* se denomina **peso *w(p)*** a la cantidad de '1‘s dentro del patrón.

Ejemplo

- Dados:
  - p = 01001010
  - q = 10001011
- Calcular los pesos:
  - w(p) = 3 y w(q) = 4
- Calcular la distancia de Hamming
  - d(p,q) = 3

Distancia mínima

- Dado un código C compuesto por patrones de bits de longitud fija, se denomina distancia mínima a la menor distancia entre dos patrones cualquiera no idénticos tomados de C.
- Para determinar la distancia mínima es necesario calcular la distancia entre cada patrón con todos los restantes del código.

Ejemplo

- Supongamos que un código se compone de los siguientes patrones que codifican cuatro caracteres:
  - A: 00000
  - B: 11100
  - C: 00111
  - D: 11011
- ¿Cuál es la distancia mínima de este código?
  - 3 bits.

Ejemplo

- Suponiendo que el transmisor envía el caracter D (11011) pero el receptor recibe el patrón 11000.
  - ¿Cuántos errores a nivel bits se produjeron?
  - ¿El receptor puede detectar el error?
  - ¿Hasta cuántos bits de error se puede detectar con un código con distancia mínima m?

Codificación de canal

- Se plantean dos estrategias:
  - **Detección de errores:** agregar información redundante para detectar los errores y pedir retransmisión de los datos.
  - **Corrección de errores:** agregar información redundante que, además de detectar los cambios, permiten la corrección del error por estadística y sin retransmisión.
- En ambos casos, el receptor debe recalcular la información redundante y compararla con la recibida.
- En ambos casos hay una relación de compromiso entre el *overhead* agregado y las capacidades de detección y corrección obtenidas.

Detección de errores

- Cuando se producen errores en una cantidad de bits menor a la distancia de Hamming del código utilizado siempre son detectados.
- Cuando se producen errores en una cantidad de bits mayor o igual a la distancia de Hamming del código utilizado pueden o no ser detectados.

Detección de errores - Paridad

- Es el esquema más simple de detección de errores.
- Se agrega un bit adicional por cada n bits de información (típicamente 8 bits).
- El bit agregado toma el valor adecuado para mantener la cantidad de '1's (peso) en un número par (o impar).
- Posee una distancia mínima de Hamming igual a 2 bits.
  - Puede detectar errores de 1 bit, o de cualquier cantidad impar de bits.
  - No detecta alteraciones en cantidades pares de bits.

Detección de errores - VRC y LRC

- Para mejorar la detección se puede utilizar el cálculo de paridad en dos dimensiones:
  - **VRC:** Se calcula la paridad en forma "vertical“
    - También conocido por TRC, por Transversal.
    - Es equivalente a la paridad normal.
    - Permite detectar un error en un bloque, pero no se sabe en qué parte del bloque.
  - **LRC:** Se calcula la paridad en forma longitudinal y se transmite como un caracter adicional al final del bloque de datos.
    - Permite detectar un error en una parte del bloque, pero no se sabe de qué bloque.
- La distancia mínima del código obtenido es igual a 3 bits.
- Si se usan ambos, es posible identificar exactamente el error ocurrido, siempre que no haya más de 1 error en cada bloque.

Bloque de datos con VRC y LRC

P	B7	B6	B5	B4	B3	B2	B1	B0	
1	0	0	1	0	1	0	0	0	DATOS
1	1	1	0	1	0	1	0	0	
0	1	0	0	0	0	1	1	0	
1	0	0	1	0	1	0	0	0	
0	1	1	1	1	0	1	1	1	
0	1	0	0	1	0	1	0	0	
1	0	0	1	0	1	0	0	0	
1	1	0	0	1	1	1	1	1	
0	1	1	0	1	1	1	0	0	
1	0	0	0	0	1	0	0	1	
1	1	0	1	0	1	1	0	0	LRC

Error detectado corregible

P	B7	B6	B5	B4	B3	B2	B1	B0	
1	0	0	1	0	1	0	0	0	DATOS
1	1	1	0	1	0	1	0	0	
0	1	0	0	0	0	1	1	0	
1	0	0	1	0	1	0	0	0	
0	1	1	1	1	0	1	1	1	
0	1	0	0	1	1	1	0	0	
1	0	0	1	0	1	0	0	0	
1	1	0	0	1	1	1	1	1	
0	1	1	0	1	1	1	0	0	
1	0	0	0	0	1	0	0	1	
1	1	0	1	0	1	1	0	0	LRC

Error detectado

P	B7	B6	B5	B4	B3	B2	B1	B0	
1	0	0	1	0	1	0	0	0	DATOS
1	1	1	0	1	0	1	0	0	
0	1	0	0	0	0	1	1	0	
1	0	0	1	0	1	0	0	0	
0	1	1	1	1	0	1	1	1	
0	1	0	0	0	1	1	0	0	
1	0	0	1	0	1	0	0	0	
1	1	0	0	1	1	1	1	1	
0	1	1	0	1	1	1	0	0	
1	0	0	0	0	1	0	0	1	
1	1	0	1	0	1	1	0	0	LRC

Error no detectado

P	B7	B6	B5	B4	B3	B2	B1	B0	
1	0	0	1	0	1	0	0	0	DATOS
1	1	1	0	1	0	1	0	0	
0	1	0	0	0	0	1	1	0	
1	0	0	1	0	1	0	0	0	
0	1	1	1	1	0	1	1	1	
0	1	0	0	0	1	1	0	0	
1	0	0	1	1	0	0	0	0	
1	1	0	0	1	1	1	1	1	
0	1	1	0	1	1	1	0	0	
1	0	0	0	0	1	0	0	1	
1	1	0	1	0	1	1	0	0	LRC

Detección de errores - CRC

- *Cyclic Redundancy Check.*
- Muy popular por ser simple de implementar y fácil de analizar matemáticamente.
- Muy bueno para detectar errores en ráfagas.
- Considera los datos a transmitir como un polinomio de potencias de 2 (un número entero largo).
- Se considera a los n bits del mensaje como los coeficientes de las n-1 primeras potencias de 2.
- Por ejemplo, el patrón 010011 es equivalente al polinomio $x^4 + x + 1$.

Detección de errores - CRC

- Dado el mensaje original *M(x)* y utilizando un polinomio generador *G(x)* de grado *r*.
- Se calcula el mensaje a transmitir *T(x)* como *M(x)* + *R(x)*,
  - donde *R(x)* es el resto de la división de *M(x)* en *G(x)*.
- De esta forma *T(x)* resulta un múltiplo exacto de *G(x)*.

Algoritmo para calcular el CRC

- Se agregan *r* bits (el grado del polinomio generador) a la derecha del mensaje *M(x)*.
- Se realiza la división en *G(x)* para obtener *R(x)*.
- Se concatena *M(x)* con *R(x)* para obtener *T(x)*.

Cálculo del CRC

M(x) 1 1 0 1 0 1 1 0 1 1 1 0 0 0 0

G(x) 1 0 0 1 1

0 1 0 0 1 1

1 0 0 1 1

0 0 0 0 0 1 0 1 1 1

1 0 0 1 1

0 0 1 0 0 0 0

1 0 0 1 1

0 0 0 1 1 0 0 R(x)

Verificación del CRC

Características del algoritmo CRC

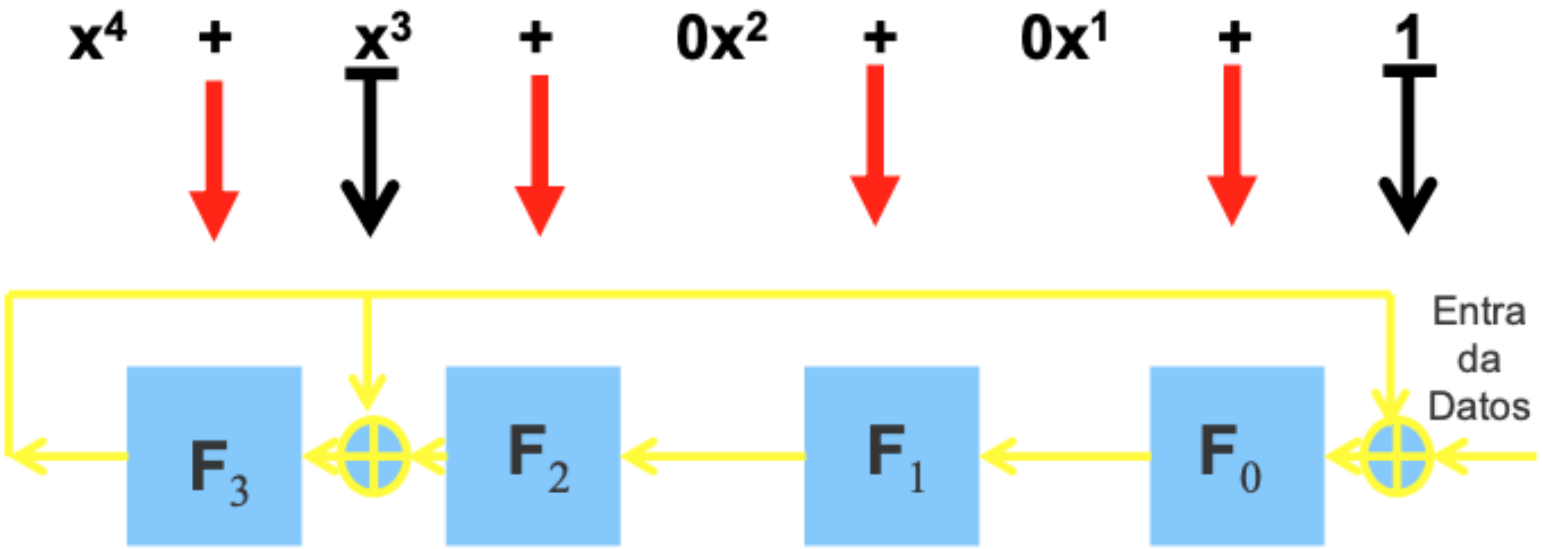
- El algoritmo permite calcular los bits que se deben agregar al mensaje original.
  - Un CRC de n bits agrega n bits al mensaje original.
- El mismo algoritmo permite verificar si el mensaje recibido contiene o no errores.
- El cálculo se puede realizar a medida que se transmiten o se reciben los bits.

Circuitos generadores de CRC

- Se utilizan tantos registros de desplazamiento como el orden r del polinomio generador.
- Se utilizan tantas compuertas XOR como '1's tenga el polinomio generador.
- Los registros se ubican en cada signo + del polinomio generador y las XOR a la derecha del registro de desplazamiento que tiene un 1, excepto en el de mayor grado.

Circuitos generadores de CRC

- Ejemplo: $G(x) = x^4+x^3+1 = 11001$



Polinomio generador de CRC

- El polinomio generador debe ser elegido con cuidado porque las capacidades de detección varían en función de las características del mismo.
- Para cada tamaño de CRC, puede haber múltiples polinomios generadores, siendo los más comunes:
  - CRC-12 = $x^{12}+x^{11}+x^3+x^2+x+1$
  - CRC-16 = $x^{16}+x^{15}+x^2+1$
  - CRC-CCITT = $x^{16}+x^{12}+x^5+1$
  - CRC-32 = $x^{32}+x^{26}+x^{23}+x^{22}+x^{16}+x^{12}+x^{11}+x^{10}+\ldots$ $\ldots+x^8+x^7+x^{5+}x^4+x^2+x+1$

  - El esquema de paridad par visto vendría a ser un CRC-1 = x+1.
  - El esquema LRC visto vendría a ser un CRC-8 = x^8+1.

Capacidades del CRC

- Analizar la capacidad de detección del CRC no es trivial.
- Detecta el 100% de las ráfagas de longitud menor o igual al orden del polinomio generador.
- Detecta la mayoría de las ráfagas de errores mayores al orden del polinomio generador.
  - Aproximadamente $(1-2^{-n})$.

Capacidades del CRC

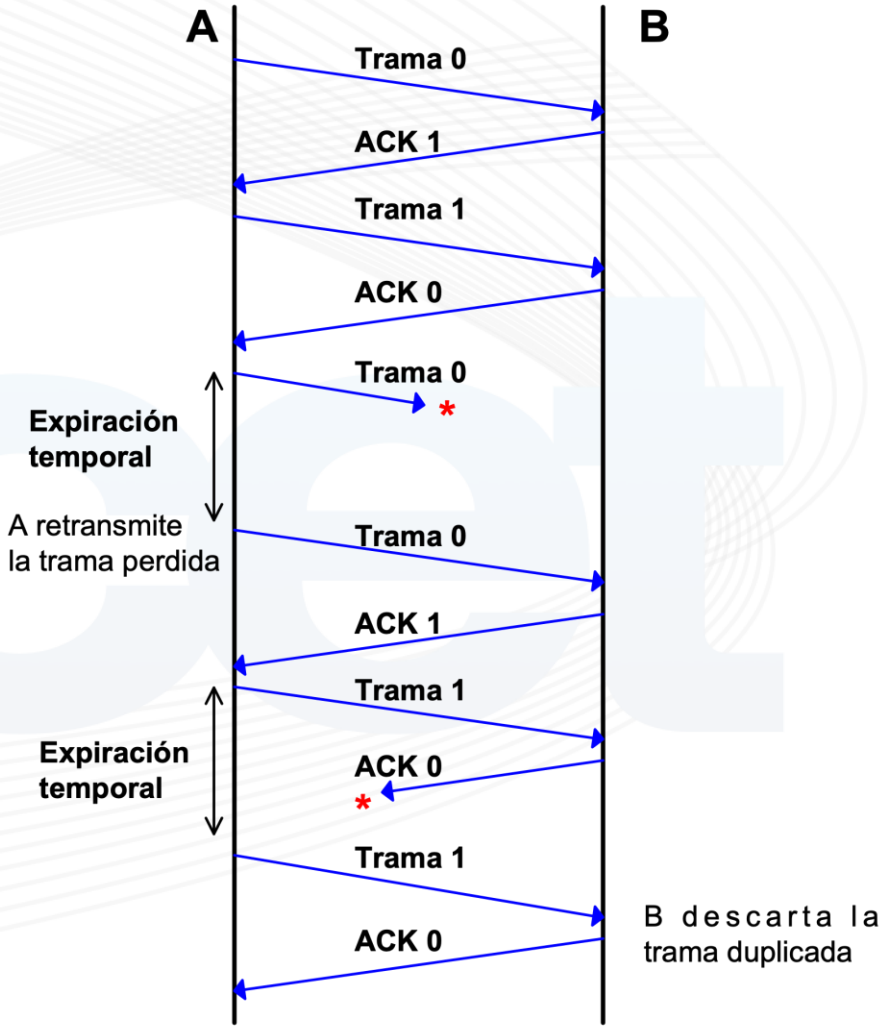
- CRC-12 detecta el 100% de errores en ráfagas de 12 bits o menos, el 99,91% de las ráfagas de 13 bits y el 99,96% del resto de ráfagas de error.
- CRC-CCITT detecta el 100% de errores en ráfagas de 16 bits o menos, el 99,994% de las ráfagas de 17 bits y el 99,997% del resto de ráfagas de error.

Métodos de retransmisión

- Cuando el receptor detecta que los datos recibidos son erróneos, una opción que tiene es pedir su retransmisión.
- El método de retransmisión más común se denomina **ARQ (*Automatic Repeat Request*)**:
  - Si el receptor recibe el mensaje sin errores, envía al transmisor un mensaje de *acknowledgment* (**ACK**).
  - Si el receptor recibe el mensaje con errores, envía al transmisor un *acknowledgment* negativo (**NACK**).
  - Entonces, si el transmisor recibe un NACK o no recibe un ACK después de un cierto tiempo de enviado el mensaje, lo reenvía.
- Este método es apropiado en casos en los que la capacidad del canal es muy variable o desconocida.
  - Como por ejemplo, en Internet.
- Hay tres variantes de ARQ.

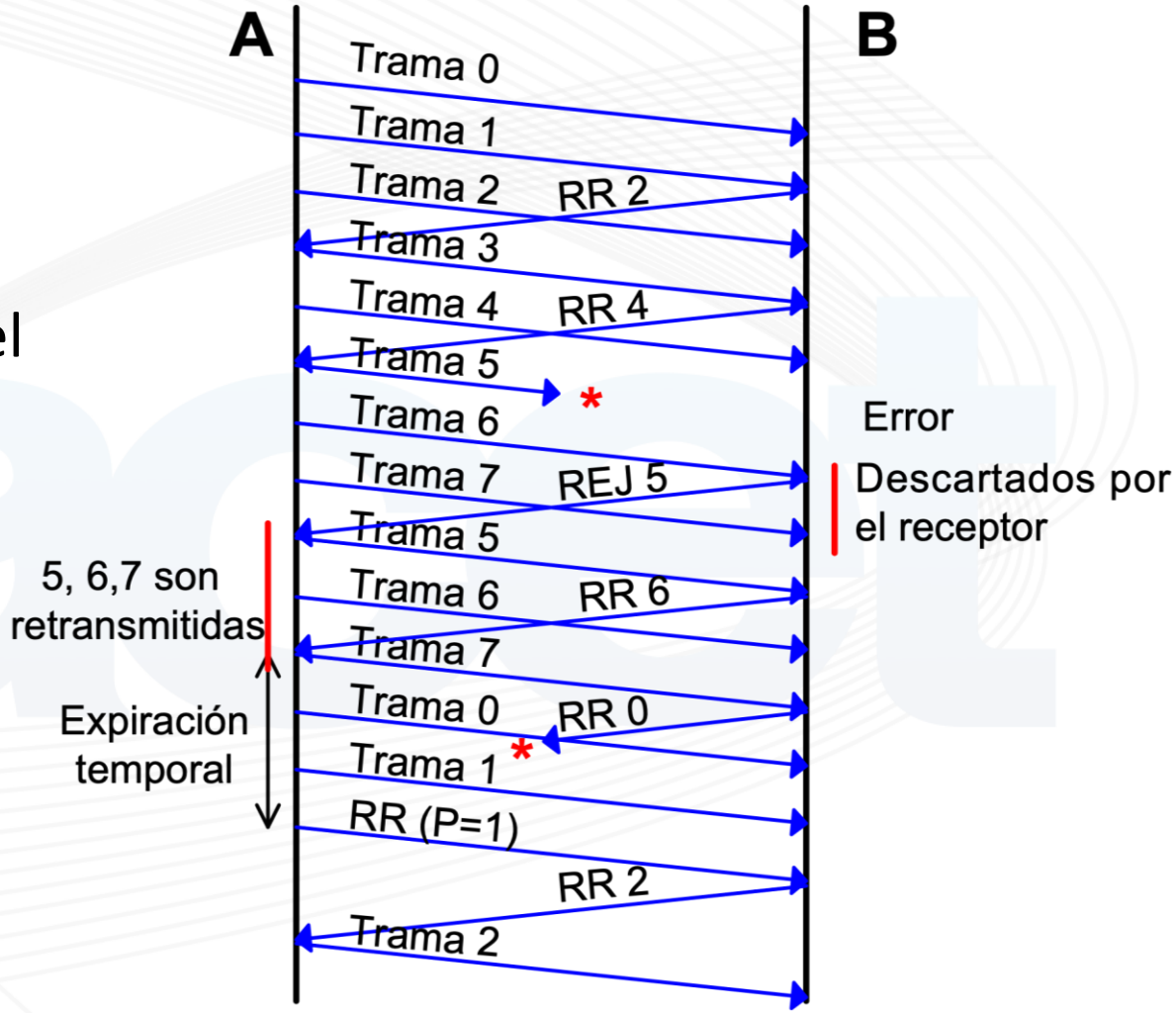
ARQ Stop-and-wait

- El transmisor envía un mensaje, y no envía el siguiente hasta que no recibe un ACK del receptor.
- Si no recibe el ACK luego de un cierto tiempo (*timeout*), reenvía el mismo mensaje.
- Problemas:
  - Se pierde el ACK, y el receptor recibe 2 veces el mismo mensaje.
  - La latencia de la transmisión es mayor que el timeout, y el receptor eventualmente recibe 2 veces el mismo mensaje.



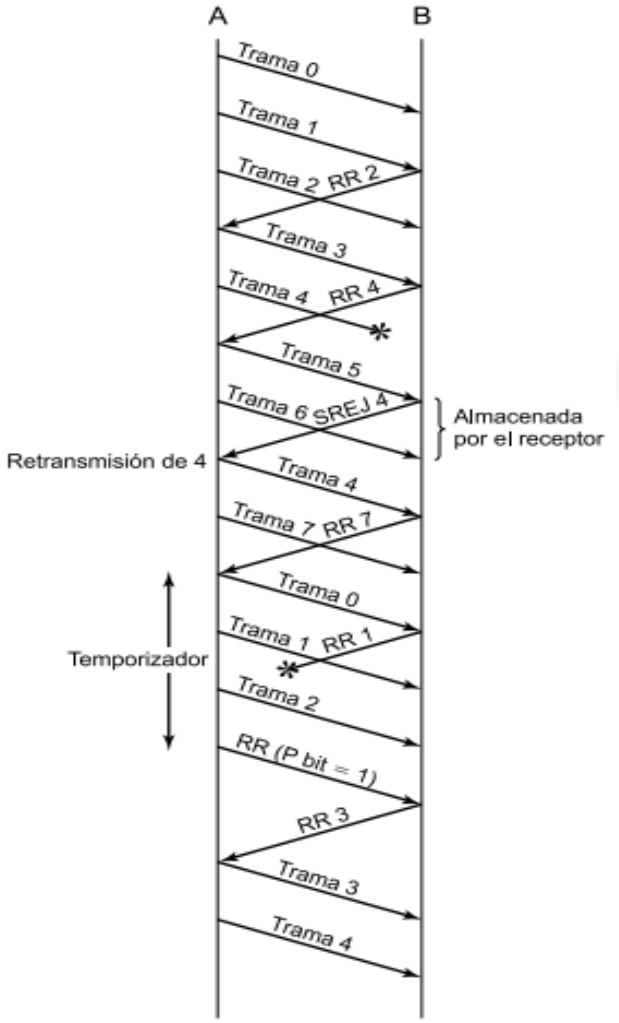
ARQ Go-back-N

- Usado por el protocolo TCP.
- El transmisor puede mandar N mensajes sin esperar el ACK del receptor.
- El receptor envía un ACK del último mensaje recibido correctamente.
- Es una mejora sobre el anterior, pero tiene los mismos problemas.



ARQ Selective Repeat

- El transmisor puede mandar N mensajes sin esperar el ACK del receptor.
- El receptor puede pedir reenvío de un mensaje en particular, el cual será retransmitido (pero solamente ese).
  - Por lo tanto, acepta recibir mensajes en cualquier orden.



Corrección de errores

- Métodos como ARQ tienen desventajas:
  - Requiere un canal secundario para el envío de mensajes ACK.
  - Aumenta las latencias, por las retransmisiones.
  - Genera una mayor congestión del medio de transmisión.
- Ciertas aplicaciones que requieren baja latencia, no pueden tolerar métodos como ARQ.
  - Como llamadas o videoconferencias.
- Sería ideal que el receptor no necesite pedir la retransmisión, sino que pueda corregir el error detectado.
  - Esto se conoce usualmente como **FEC (*Forward Error Correction*)**.
  - Usados no sólo para transmisión de datos, sino para almacenamiento (memorias RAM, memorias FLASH y discos).

Códigos de FEC

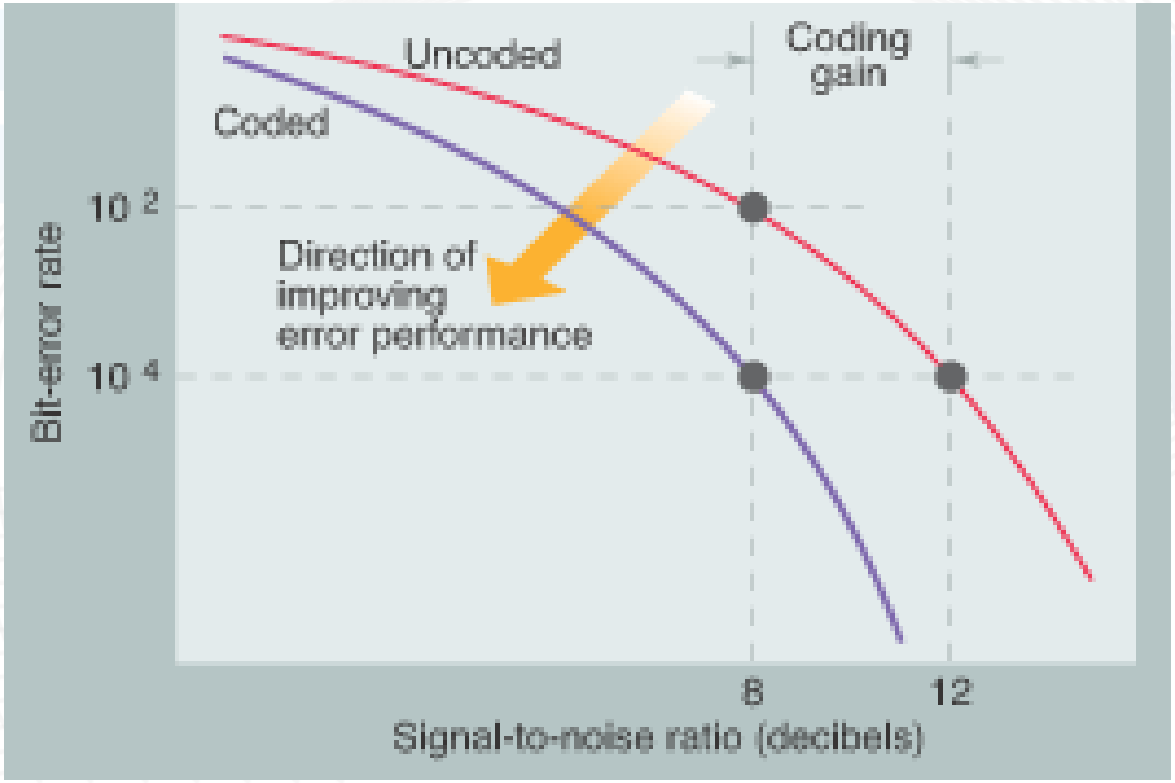
- Existen dos tipos básicos de códigos:
  - Códigos por bloques lineales: el procesamiento se hace por bloques de bits de un tamaño dado.
    - Códigos de repetición.
    - Códigos de Hamming.
    - Códigos de paridad multidimensional.
    - Códigos tipo Reed-Solomon.
    - Códigos Turbo.
  - Códigos convolucionales: el procesamiento se hace por bits, en secuencias de cualquier tamaño.
    - Óptimos para implementaciones en hardware.
    - Algoritmo de Viterbi.

Códigos de FEC

- Se caracterizan por:
  - Capacidad de corrección, o sea cúantos errores pueden corregir.
    - Un código con una distancia mínima de Hamming ***d*** puede detectar como máximo *d-1* errores y puede corregir *(d-1)/2* errores.
  - Eficiencia o tasa $\eta = k/n$
    - Relación entre símbolos útiles transmitidos (*k*) y los realmente transmitidos (*n*).
    - Esto también se conoce como ***code rate***.
    - La redundancia sería *n – k* bits o *(n – k) / n* en valor porcentual.
    - Cuanto mayor sea la redundancia, menor es la eficiencia, pero más robusto es el código.
    - Cuanto mayor sea la redundancia, menor será la velocidad de transmisión.
  - Complejidad computacional para la codificación y decodificación.

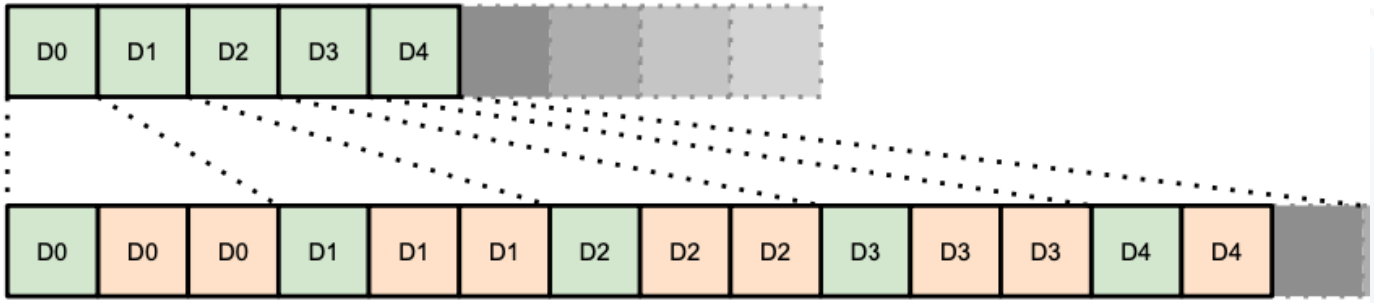
Códigos de FEC

- En general, el uso de códigos FEC permite mejorar la performance de una transmisión, dada una determinada SNR.
- O visto desde otro punto de vista, permite obtener la misma performance con una SNR menor.



Códigos de repetición

- Dada una secuencia de bits a transmitir, la idea clave es repetir cada bit un determinado número de veces *n*, que debe ser siempre impar.



- Ejemplo:
  - Para transmitir 1011, con n=3, se transmite 111 000 111 111.
  - Si se recibe 110 000 111 111, hubo un error.
- Como el receptor conoce *n*, en caso de ocurrir un error puede corregirlo por mayoría.

Códigos de repetición

- Son extremadamente simples.
- Tienen una distancia mínima de Hamming igual a n.
  - Pueden detectar errores de hasta n–1 bits.
  - Pueden corregir errores de hasta (n–1) / 2 bits.
- ¡Son muy ineficientes!
  - La tasa de datos es 1/n.

Códigos de Hamming

- El mensaje se divide en dos partes bien diferenciadas:
  - El mensaje original (de tamaño *k*).
  - Y la redundancia que se agrega (de tamaño *n*–*k*).
- Pueden detectar errores de 1 bit y de 2 bits, y corregir errores de hasta 1 bit.
  - O sea, poseen d = 3.
- Son los códigos FEC que poseen la mayor eficiencia para esa distancia de Hamming.
- Muy usados, por ejemplo en memorias RAM.

Códigos de Hamming

- Para un bloque de datos *k* se incorporan *r* bits (con r ≥ 2) de paridad para satisfacer la inecuación:
  - $k + r + 1 \leq 2^r$
- Por ejemplo para 8 bits de datos se deben agregar 4 bits de paridad.
- A este código se lo denomina (12,8) porque tiene 12 bits en total con 8 bits de datos.
  - El ejemplo de código de repetición visto, es un código de Hamming (3,1) y una paridad simple sobre un ASCII de 7 bits sería un código de Hamming (8,7).
- La redundancia son bits de paridad que se calculan sobre determinados bits de datos.
- Se reservan las posiciones potencias de dos para ubicar los bits de redundancia.
- La idea general es lograr que una XOR de todas las posiciones que contengan '1's sea igual a cero.

Códigos de Hamming (algoritmo)

- Se numeran **en binario** todas las posiciones de bits del mensaje, empezando por 1.
- Todas las posiciones que son potencias de 2 serán bits de paridad.
  - Las restantes, bits de datos.
- El primer bit de paridad, P0, se calcula sobre todas las posiciones que tengan su bit menos significativo igual a 1.
  - O sea, todas las posiciones impares.
- El segundo bit de paridad, P1, se calcula sobre todas las posiciones que tengan su segundo bit menos significativo igual a 1.
  - O sea, las posiciones 2, 3, 6, 7, etc.
- El tercer bit de paridad, P2, se calcula sobre todas las posiciones que tengan su tercer bit menos significativo igual a 1.
  - O sea, las posiciones 4 a 7, 12 a 15, etc.
- El cuarto bit de paridad, P3, se calcula sobre todas las posiciones que tengan su cuarto bit menos significativo igual a 1.

Código de Hamming (ejemplo 12,8)

P_0	P_1	D_0	P_2	D_1	D_2	D_3	P_3	D_4	D_5	D_6	D_7
P		X		X		X		X		X	
	P	X			X	X			X	X	
			P	X	X	X					X
							P	X	X	X	X
I	0	I	0	I	0	I	0	I	0	I	0
0	I	I	0	0	I	I	0	0	I	I	0
0	0	0	I	I	I	I	0	0	0	0	I
0	0	0	0	0	0	0	I	I	I	I	I

Código de Hamming (ejemplo 12,8)

- Supongamos que queremos transmitir la palabra 10011010, de 8 bits.
- Primero calculamos cuántos bits deberíamos agregar, que son 4, y los agregamos en las posiciones potencias de 2, quedando así: __1_001_1010.
- Luego calculamos las paridades. Asumiendo paridad par, el mensaje a transmitir quedaría 011100101010.

Cálculo del Síndrome

- Un código de Hamming asegura que cada bit de datos es incluido en un conjunto único de bits de paridad.
- Entonces, revisando los bits de paridad, se pueden detectar con precisión los errores.
- El patrón de errores es denominado **síndrome**.
- Usando la matriz de posiciones binarias y los datos recibidos se puede calcular el síndrome para determinar si hubo errores.
  - Si se obtiene un síndrome nulo, no se produjeron errores.
  - Un síndrome no nulo indica la posición del bit en el que se produjo el error.
  - Por ejemplo, si los bits de paridad de las posiciones 1, 2 y 8 son incorrectos, entonces hay un error en el bit ubicado en la posición 11.

Cálculo del Síndrome (ejemplo 12,8)

$$\begin{bmatrix} S_0 \\ S_1 \\ S_2 \\ S_3 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{bmatrix} \times \begin{bmatrix} P_0 \\ P_1 \\ D_0 \\ P_2 \\ D_1 \\ D_2 \\ D_3 \\ P_3 \\ D_4 \\ D_5 \\ D_6 \\ D_7 \end{bmatrix}$$

Cálculo del Síndrome (ejemplo 12,8)

- El cálculo matricial anterior es equivalente a calcular la paridad sobre los bits del mensaje recibido:

$$S_0 = P_0 \oplus D_0 \oplus D_1 \oplus D_3 \oplus D_4 \oplus D_6$$
$$S_1 = P_1 \oplus D_0 \oplus D_2 \oplus D_3 \oplus D_5 \oplus D_6$$
$$S_2 = P_2 \oplus D_1 \oplus D_2 \oplus D_3 \oplus D_7$$
$$S_3 = P_3 \oplus D_4 \oplus D_5 \oplus D_6 \oplus D_7$$

Cálculo de Hamming (12,8)

- Otro ejemplo: queremos transmitir 01010101, con paridad par:

P_0	P_1	D_0	P_2	D_1	D_2	D_3	P_3	D_4	D_5	D_6	D_7
		0		1	0	1		0	1	0	1
0		0		1		1		0		0	
	0	0			0	1			1	0	
			1	1	0	1					1
							0	0	1	0	1
0	0	0	1	1	0	1	0	0	1	0	1

Detección de error Hamming (12,8)

		P_0	P_1	D_0	P_2	D_1	D_2	D_3	P_3	D_4	D_5	D_6	D_7
		0	0	0	1	0	0	1	0	0	1	0	1
S_0	1	0		0		0		1		0		0	
S_1	0		0	0			0	1			1	0	
S_2	1				1	0	0	1					1
S_3	0								0	0	1	0	1

Códigos de Hamming - Implementación

- Una característica que ayuda mucho a la popularidad de estos códigos, es que su codificación (y decodificación) se puede implementar sencillamente en hardware.
- Al igual que con CRC, son necesarios registros de desplazamiento y compuertas XOR.
  - Cada variante de código ubicará en distintos lugares sus componentes.

Códigos Reed-Solomon

- Son excelentes para corregir errores en ráfagas.
- Por ello, muy usados:
  - CDs, DVDs, BluRays, etc.
  - Códigos de barra, QR, etc.
  - xDSL, satélites, etc.
- Parten de una idea similar a los códigos de Hamming, pero poseen una distancia igual a r +1.
  - Pueden detectar errores de hasta r bits.
  - Pueden corregir errores de hasta r/2 bits.
- Por ejemplo, un código RS común es definido como (255,223), por lo que:
  - Genera bloques de 255 bits de longitud, a partir de bloques de datos de 223 bits, aumentando 32 bits de paridad.
  - Detecta hasta errores de 31 bits, y corrige errores de hasta 16 bits.

Códigos Reed-Solomon

- Las variantes más comunes consideran los bits del mensaje a transmitir como los primeros k valores de un polinomio, el cual generan interpolando estos valores con otro polinomio generador, de grado menor que k.
  - Usualmente, se usa una interpolación de Lagrange, o una transformada discreta de Fourier.
  - No se preocupen, que no lo vamos a hacer ☺
  - Al igual que los códigos de Hamming, para la decodificación se usa el método de los síndromes, o alguno similar.
  - Al igual que los códigos de Hamming, son simples de implementar en hardware con LFSR y XOR.

Códigos convolucionales

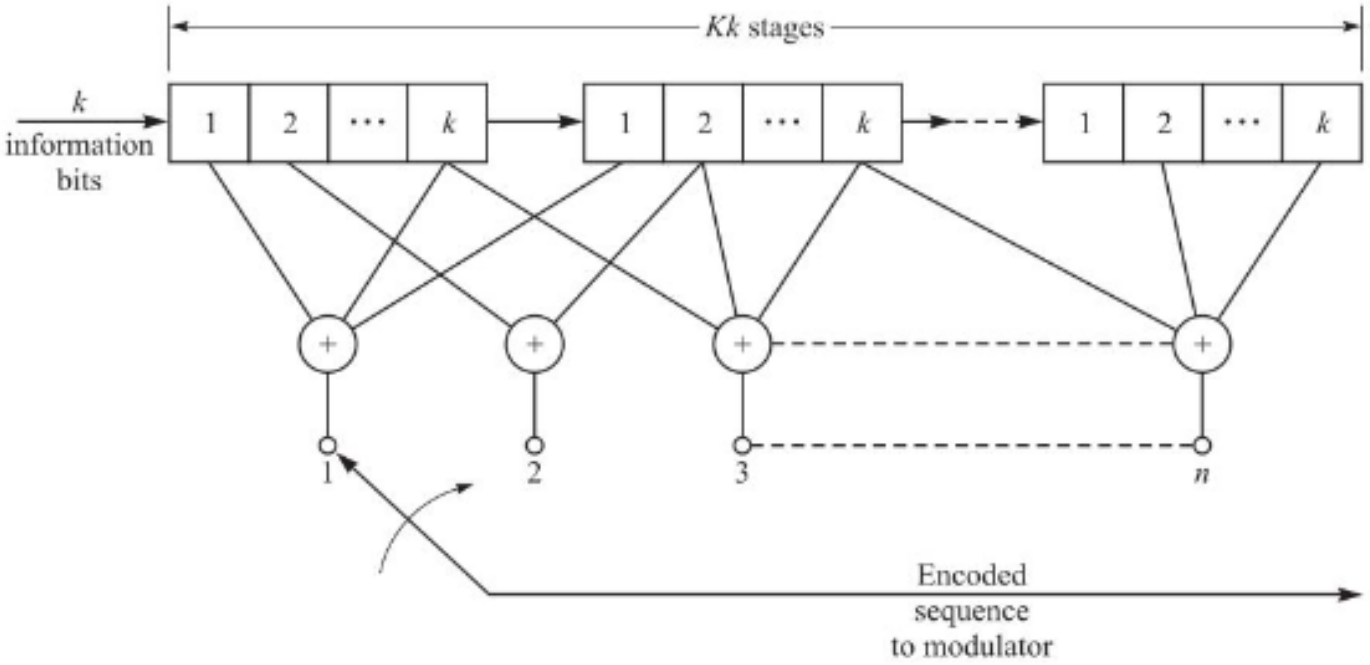
- Al igual que los códigos de bloque son lineales pero, a diferencia de ellos, éstos tienen memoria.
  - No operan sobre bloques de tamaño fijo, sino de tamaño arbitrario.
- La codificación resultante no depende sólo de los datos actuales, sino tambien de los datos anteriores.
  - Consiste en la aplicación en forma de “ventana deslizante” de una función polinomial al stream de bits, para generar bits de paridad.
- La decodificación consiste en elegir la cadena más probable dentro ciertas opciones, y se realiza en forma óptima utilizando un algoritmo conocido (algoritmo de Viterbi).

Códigos convolucionales

K : etapas

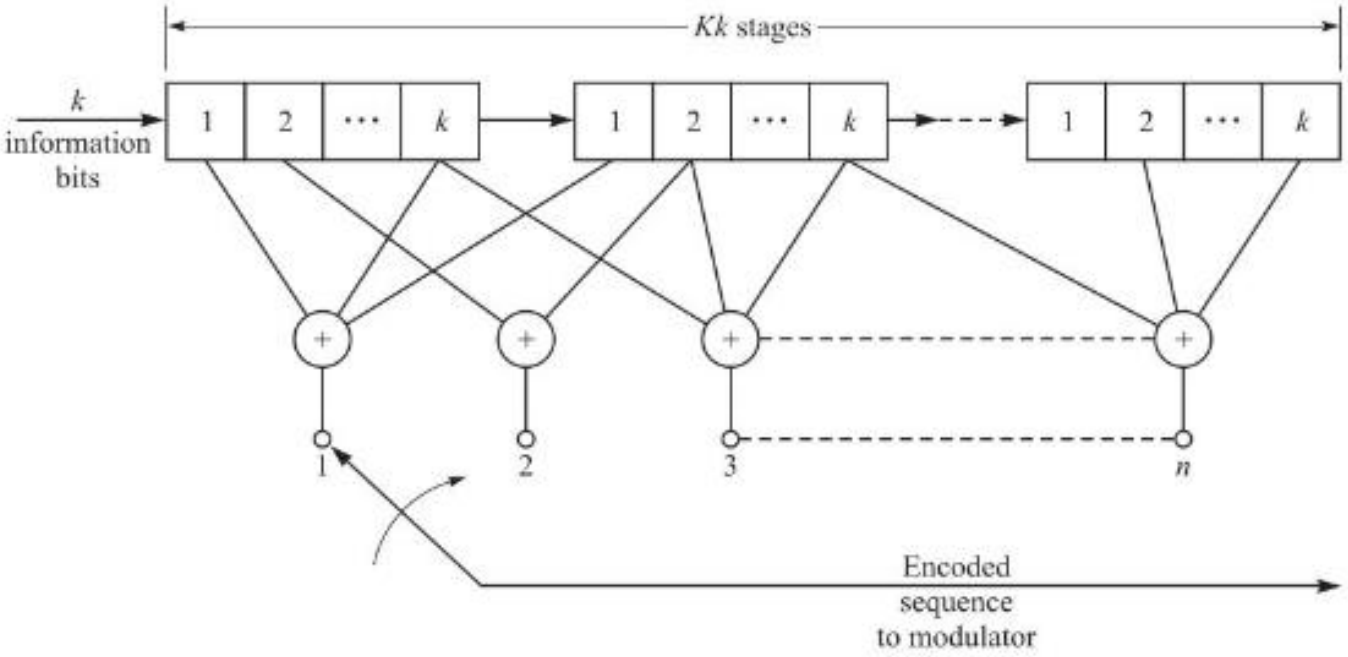
k : bits de datos

n : bits de código



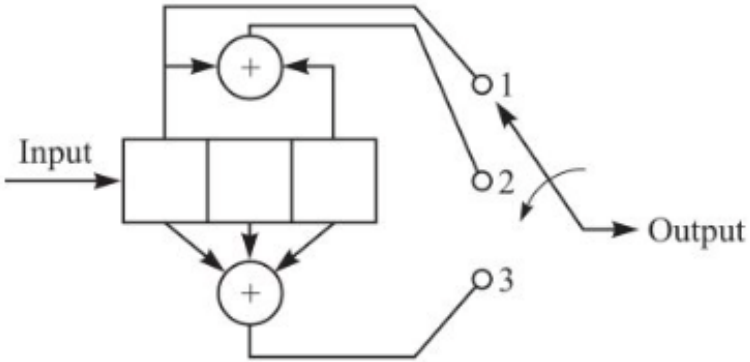
Códigos convolucionales

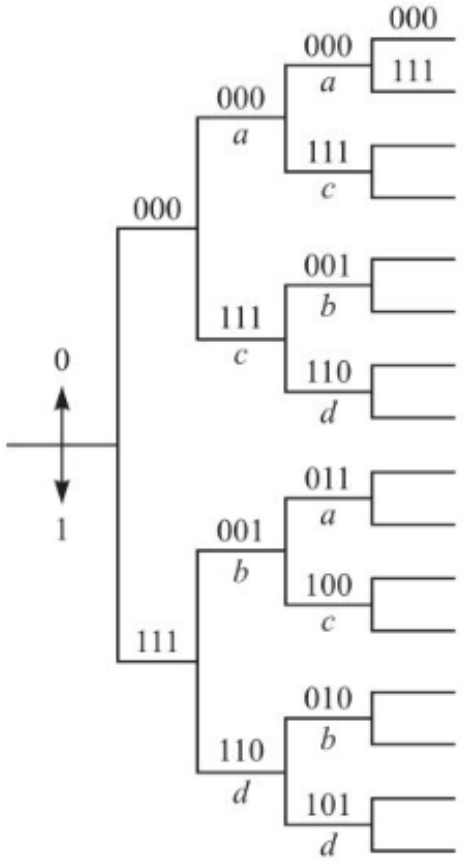
- Un codificador convolucional toma k bits de datos para generar n bits de código a partir de los K últimos datos transmitidos.
  - K representa la memoria, o la “profundidad” del codificador.



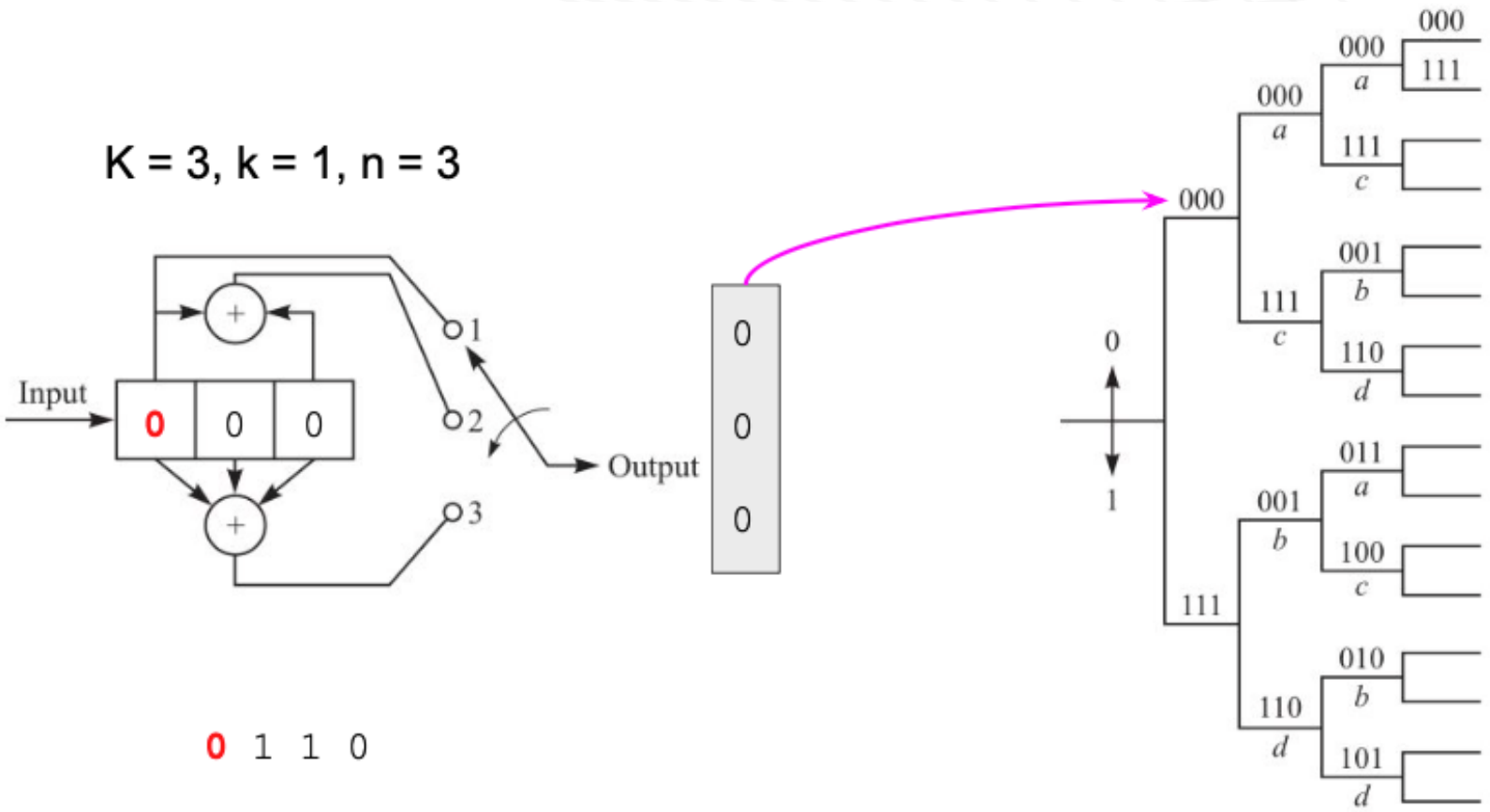
Códigos convolucionales

K = 3, k = 1, n = 3

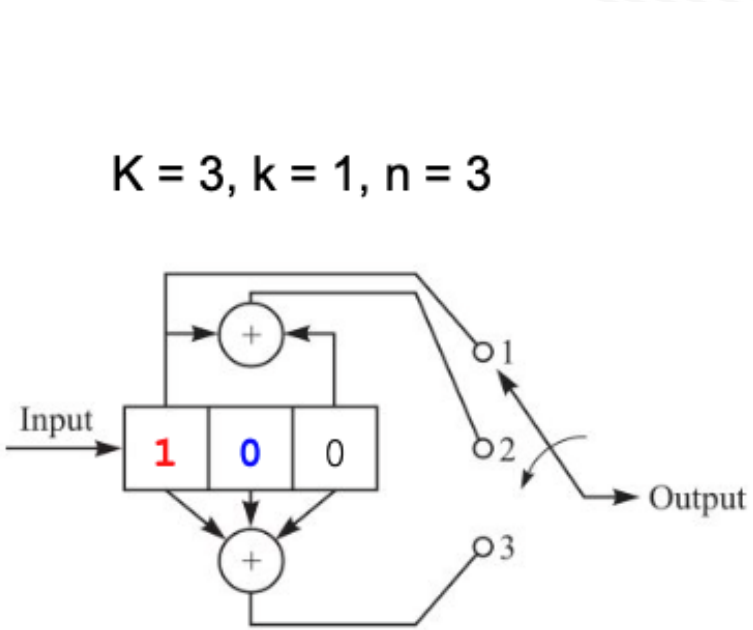


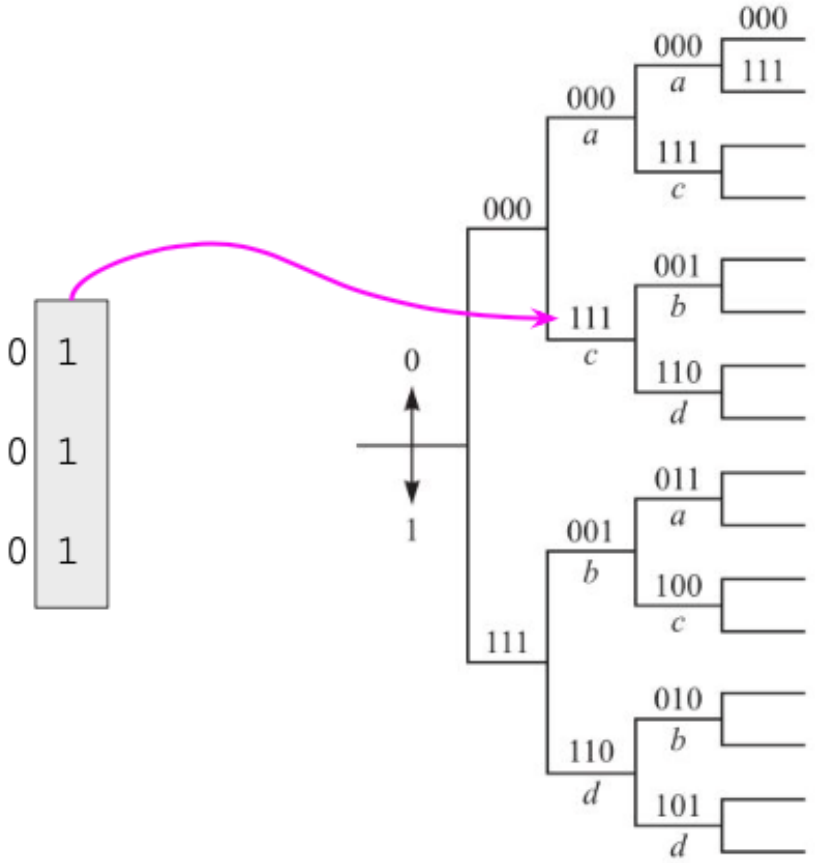


Códigos convolucionales

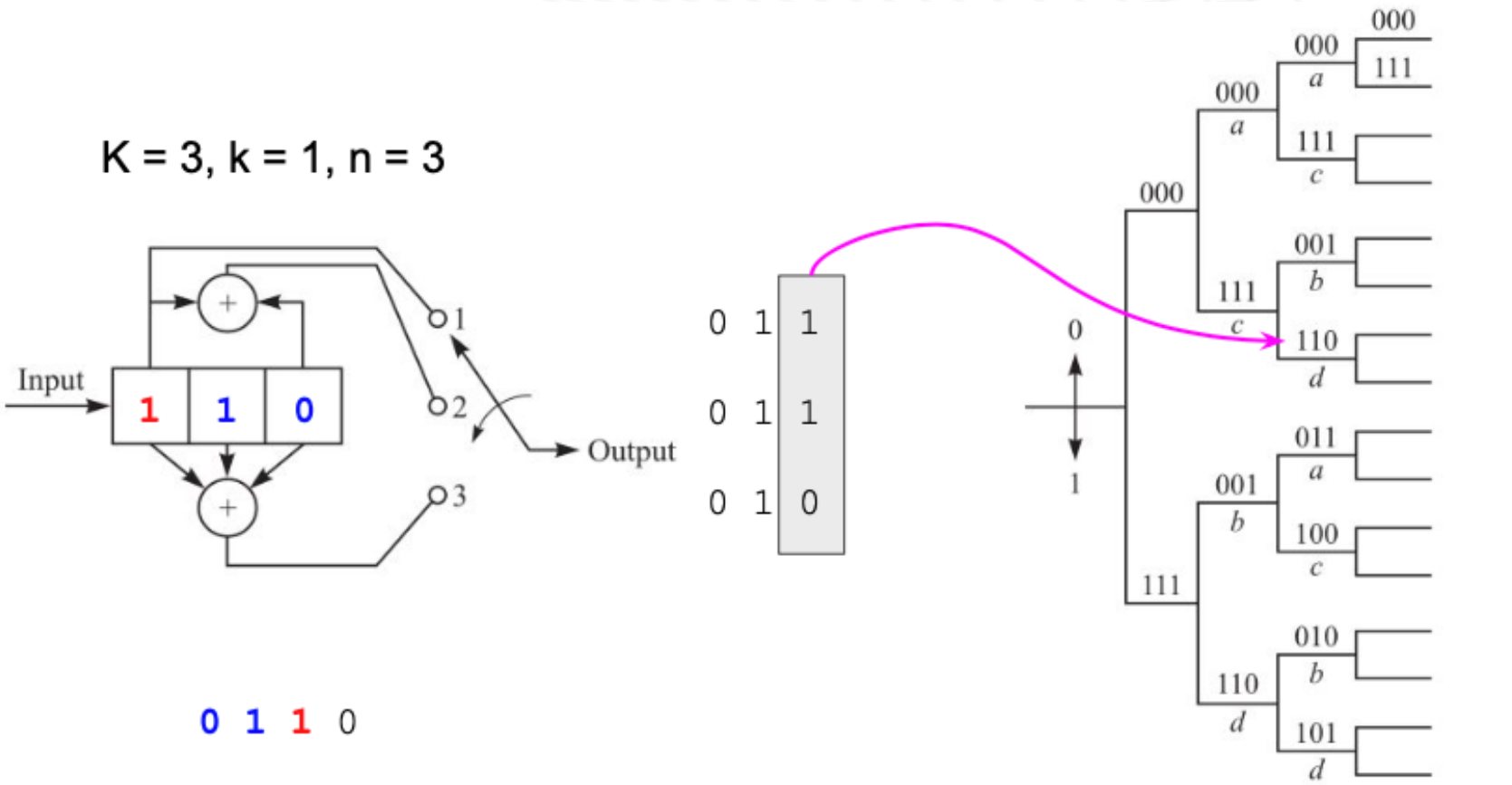


Códigos convolucionales

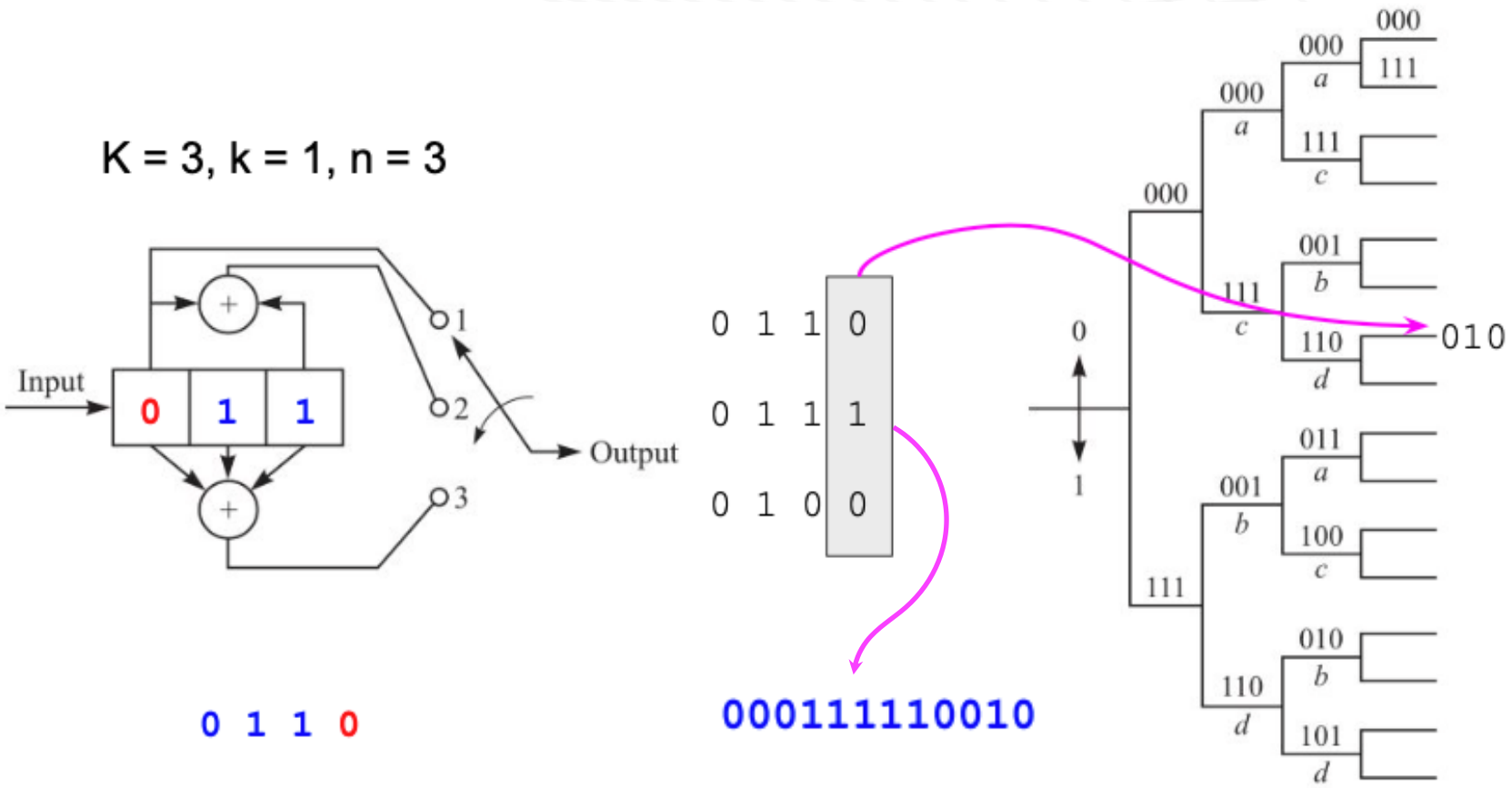




Códigos convolucionales



Códigos convolucionales

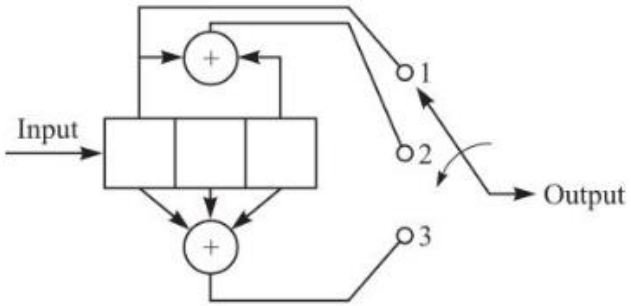


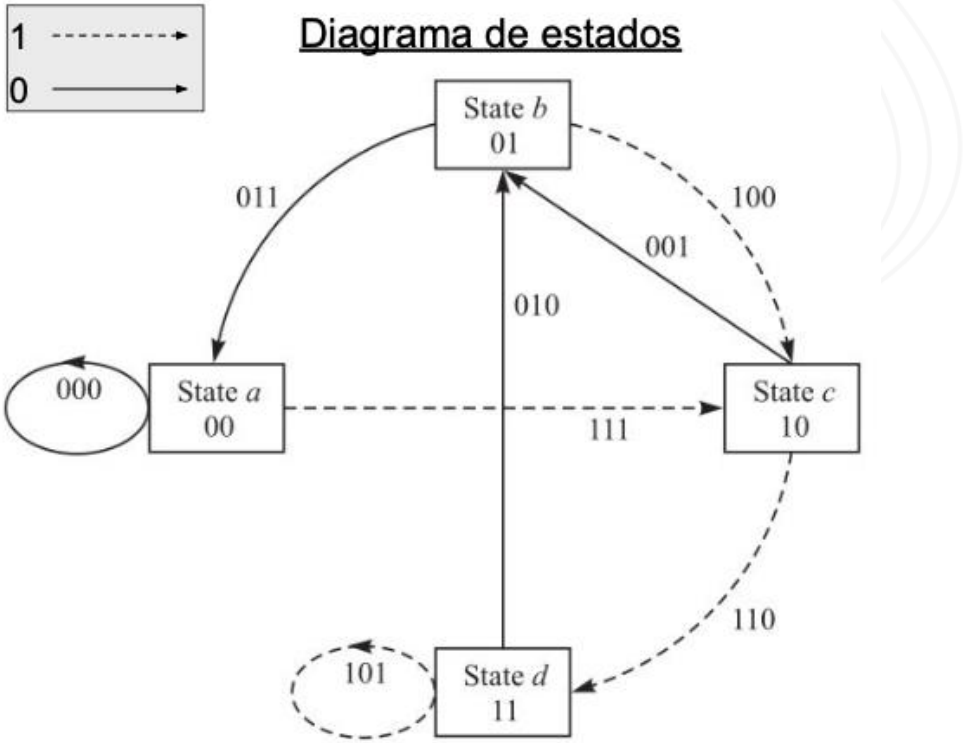
Códigos convolucionales

- Un codificador convolucional es, en esencia, una Máquina de Estados Finitos.

Códigos convolucionales:

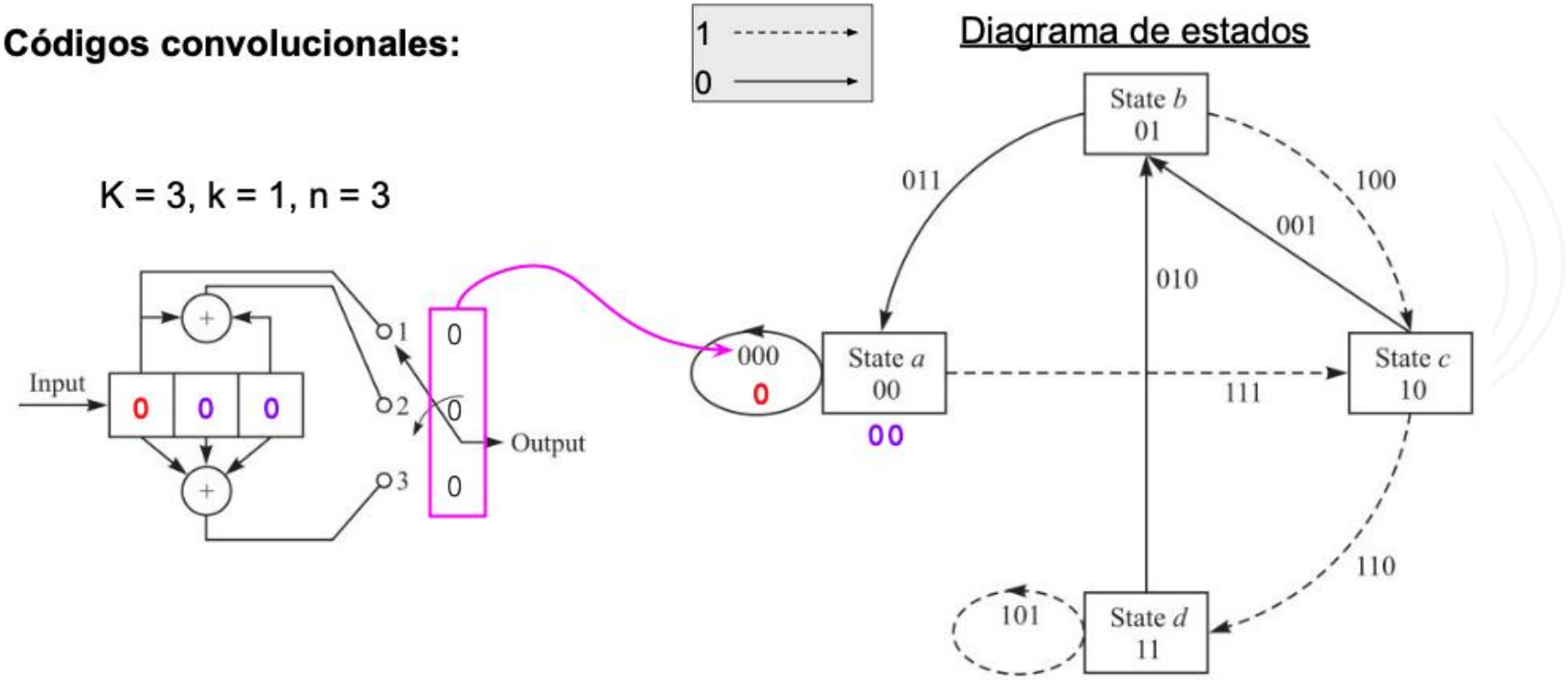
K = 3, k = 1, n = 3





Códigos convolucionales

Códigos convolucionales:



Códigos convolucionales

Códigos convolucionales:

K = 3, k = 1, n = 3

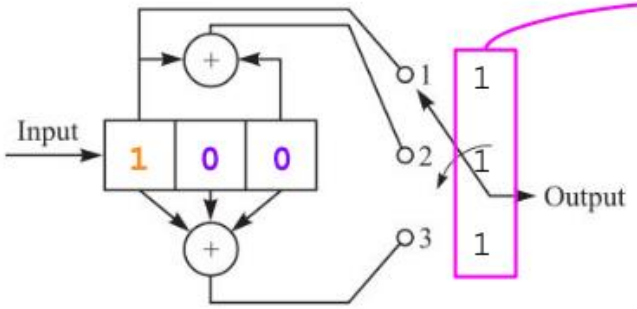
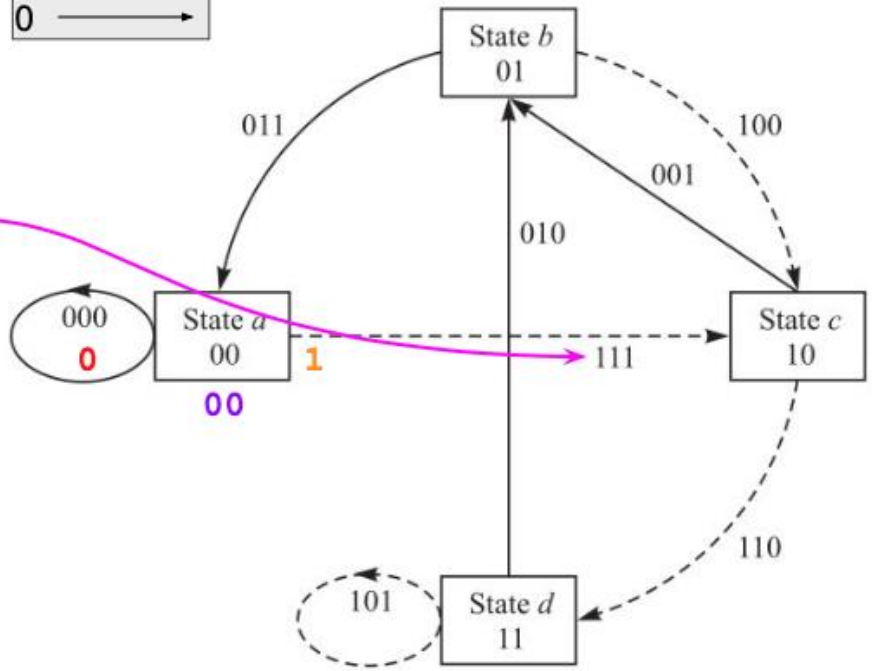


Diagrama de estados



Códigos convolucionales

Códigos convolucionales:

K = 3, k = 1, n = 3

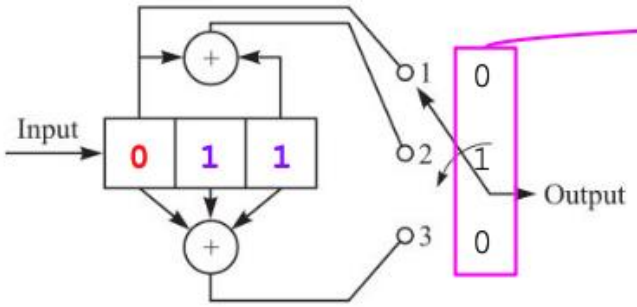
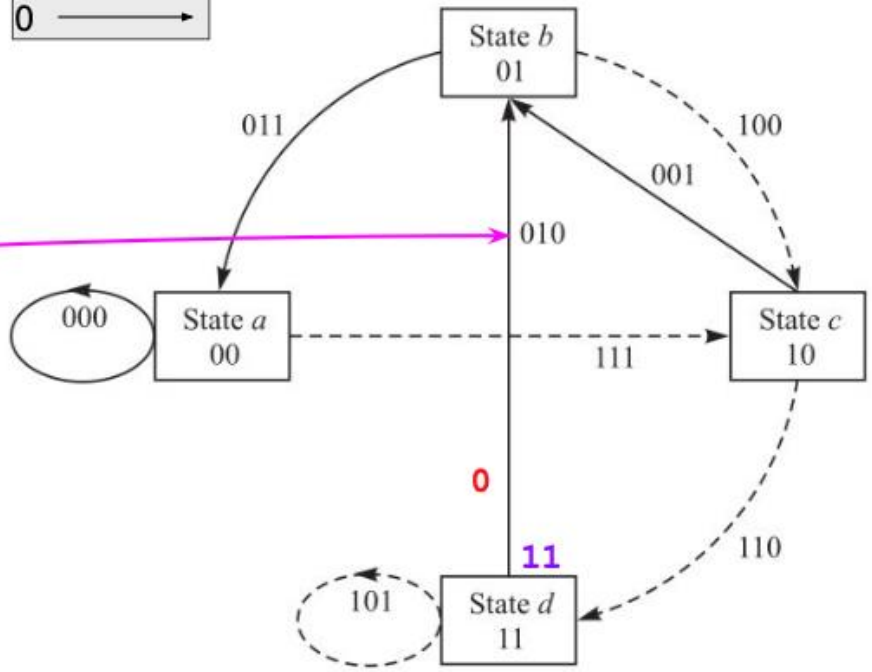


Diagrama de estados



Códigos convolucionales

Códigos convolucionales:

K = 3, k = 1, n = 3

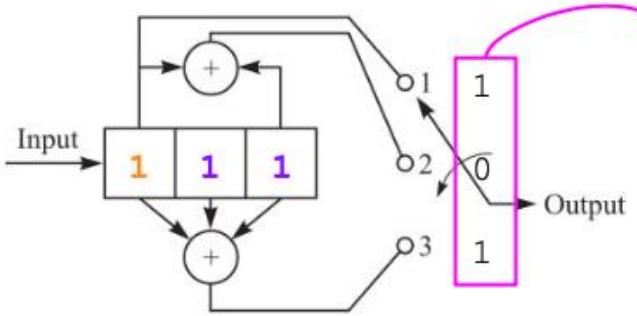
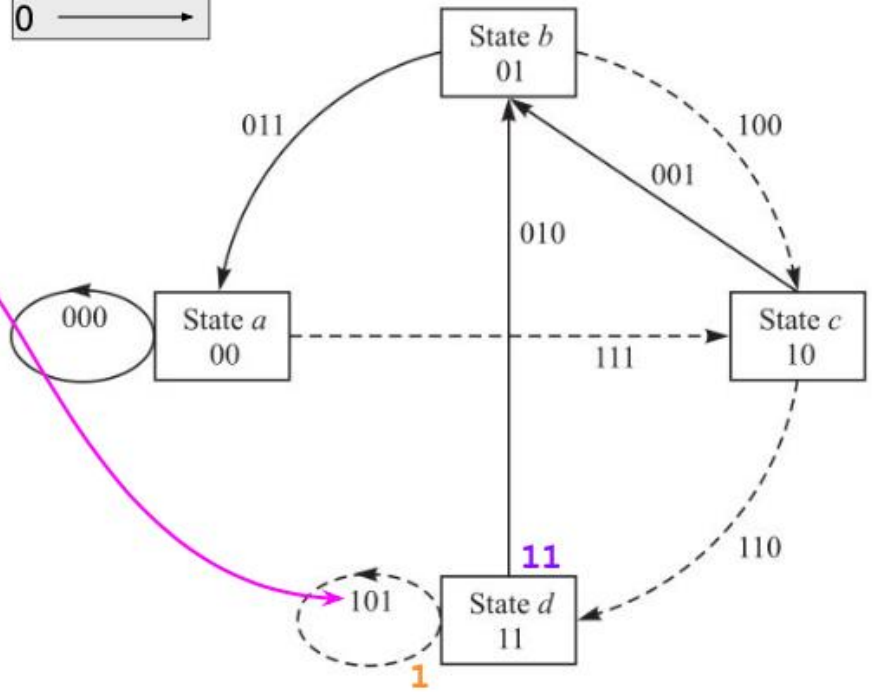


Diagrama de estados



Códigos convolucionales

Códigos convolucionales:

K = 3, k = 1, n = 3

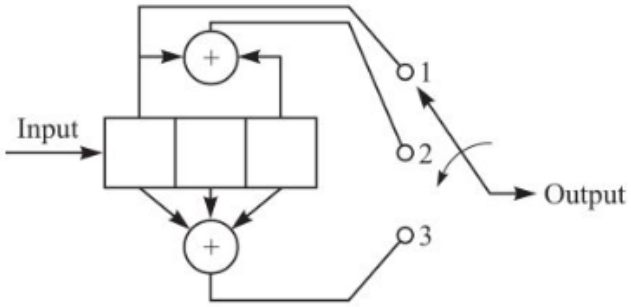
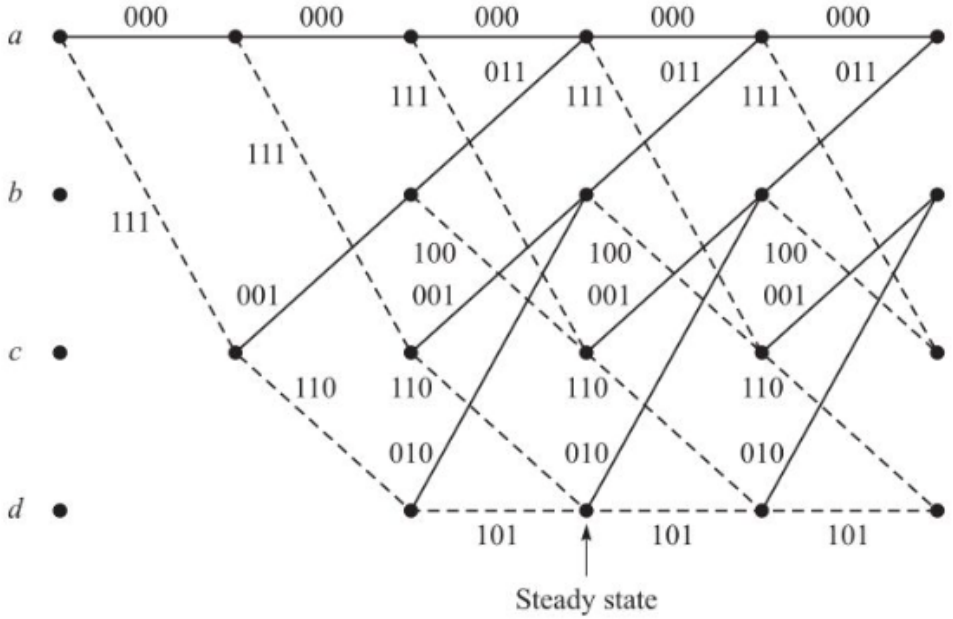
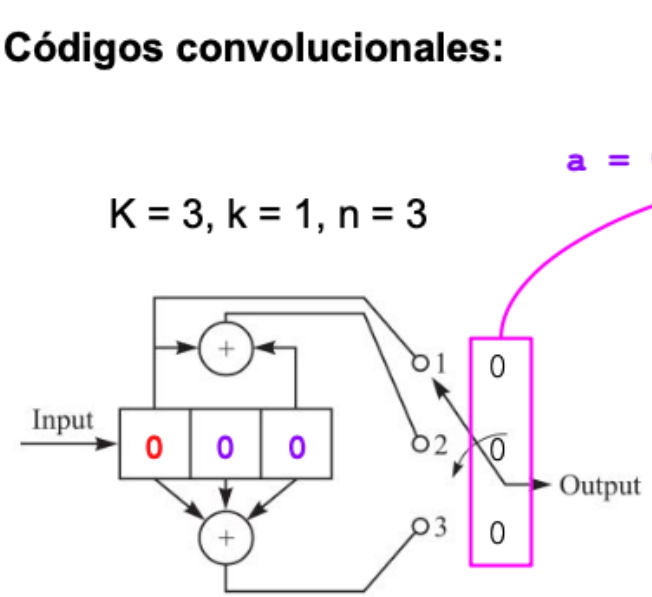


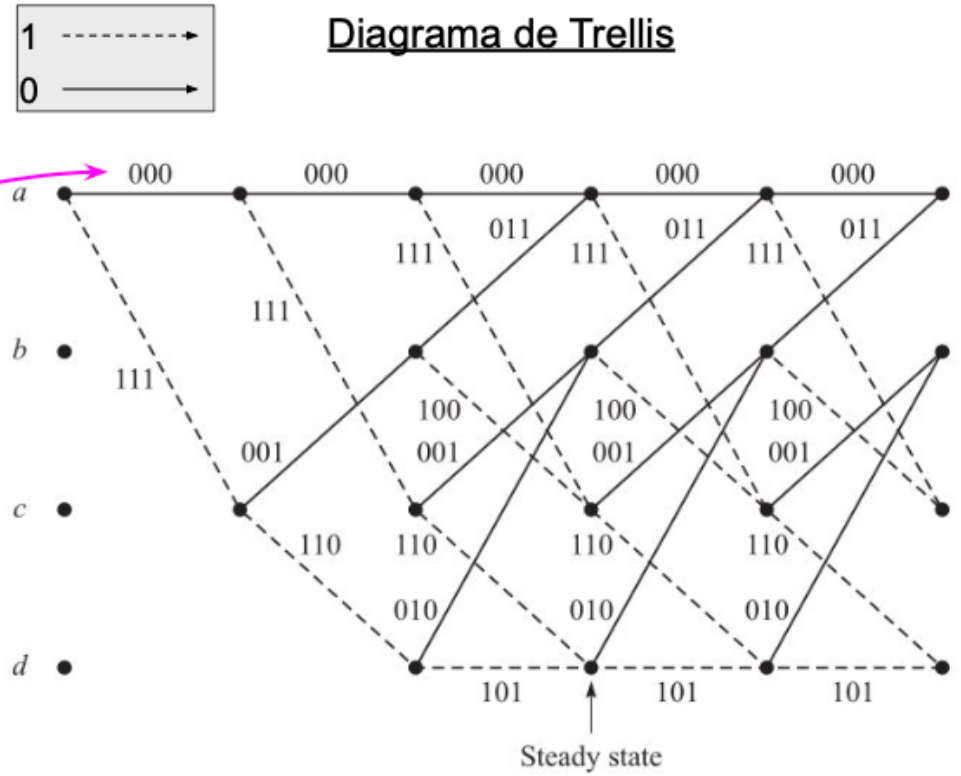
Diagrama de Trellis



Códigos convolucionales

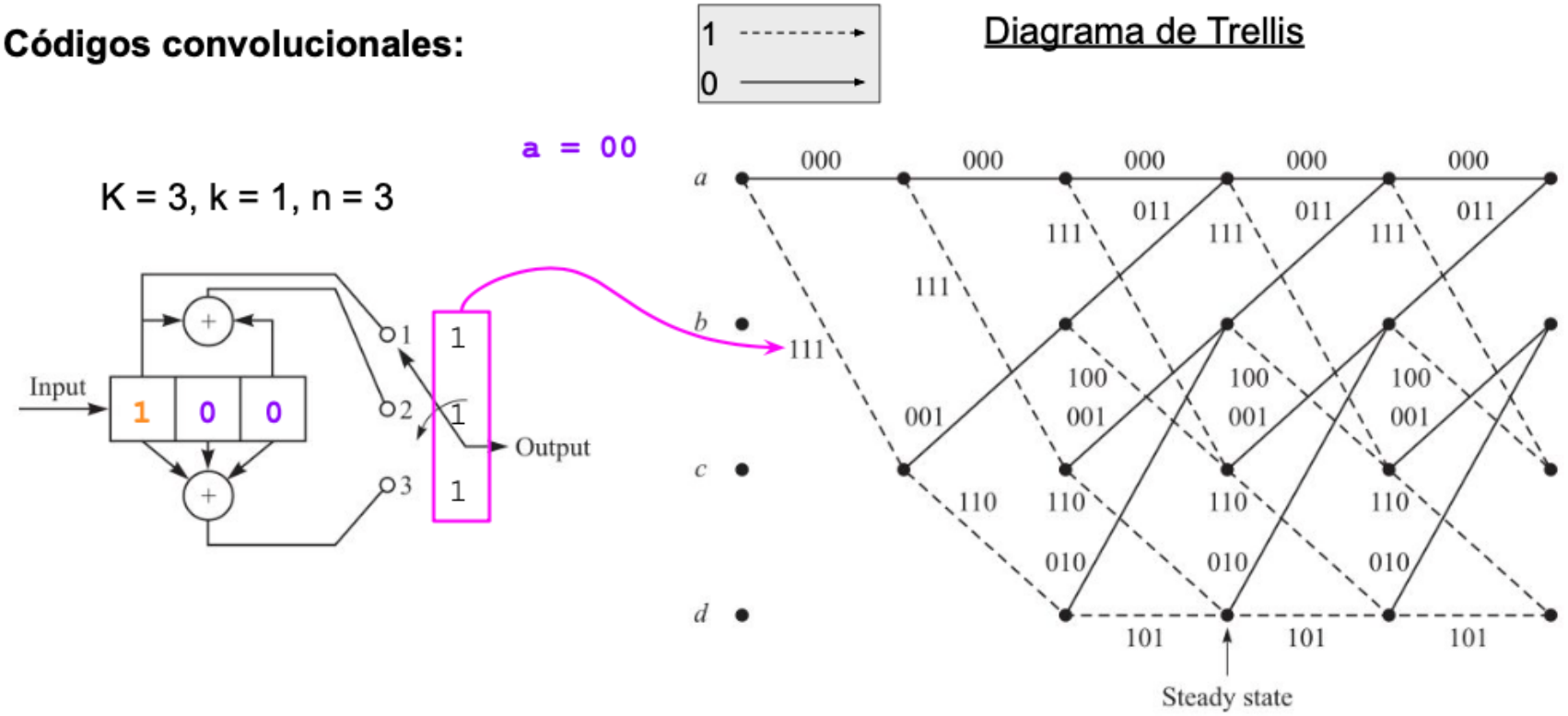
Códigos convolucionales:





Códigos convolucionales

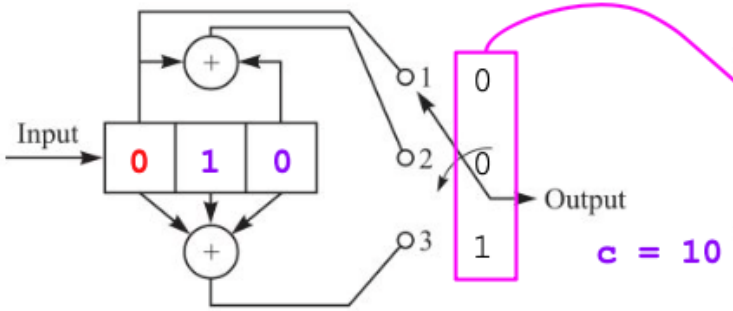
Códigos convolucionales:



Códigos convolucionales

Códigos convolucionales:

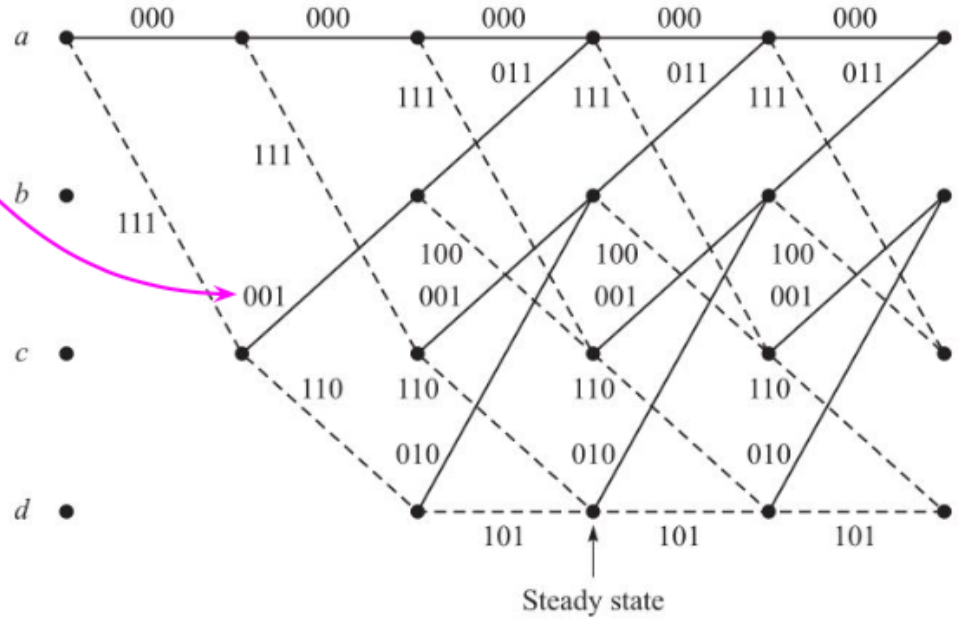
K = 3, k = 1, n = 3



1 ---------->

0 ——————>

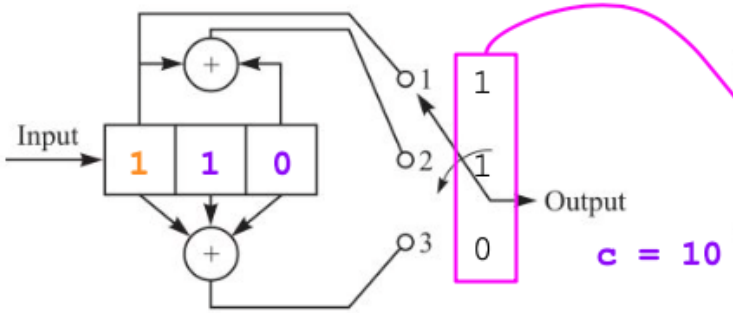
Diagrama de Trellis

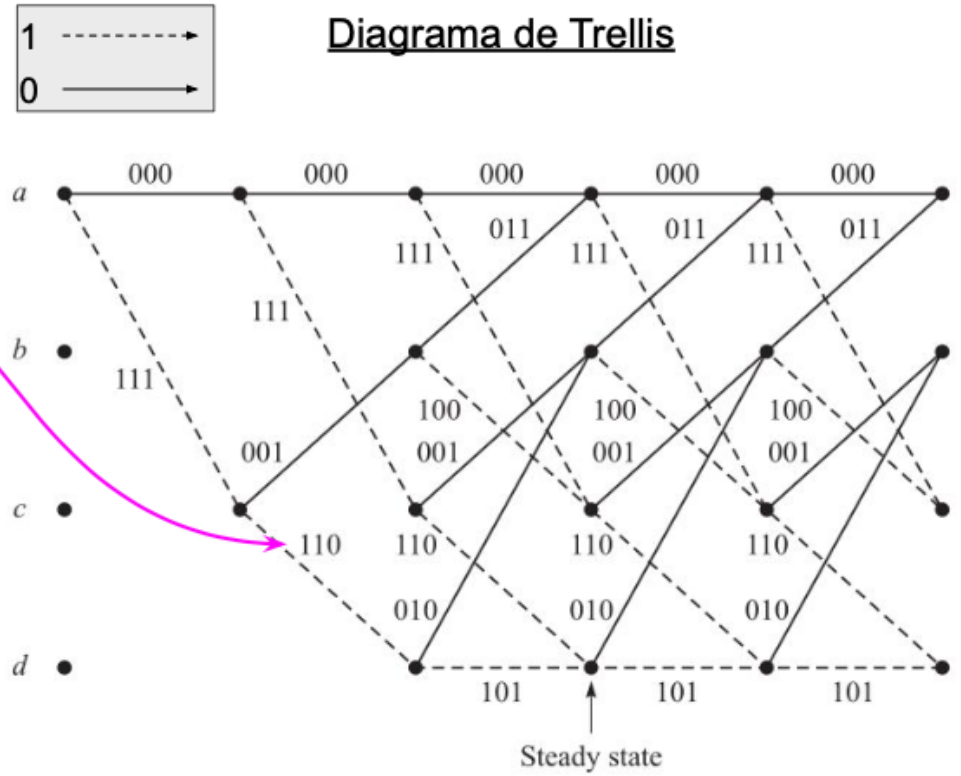


Códigos convolucionales

Códigos convolucionales:

K = 3, k = 1, n = 3





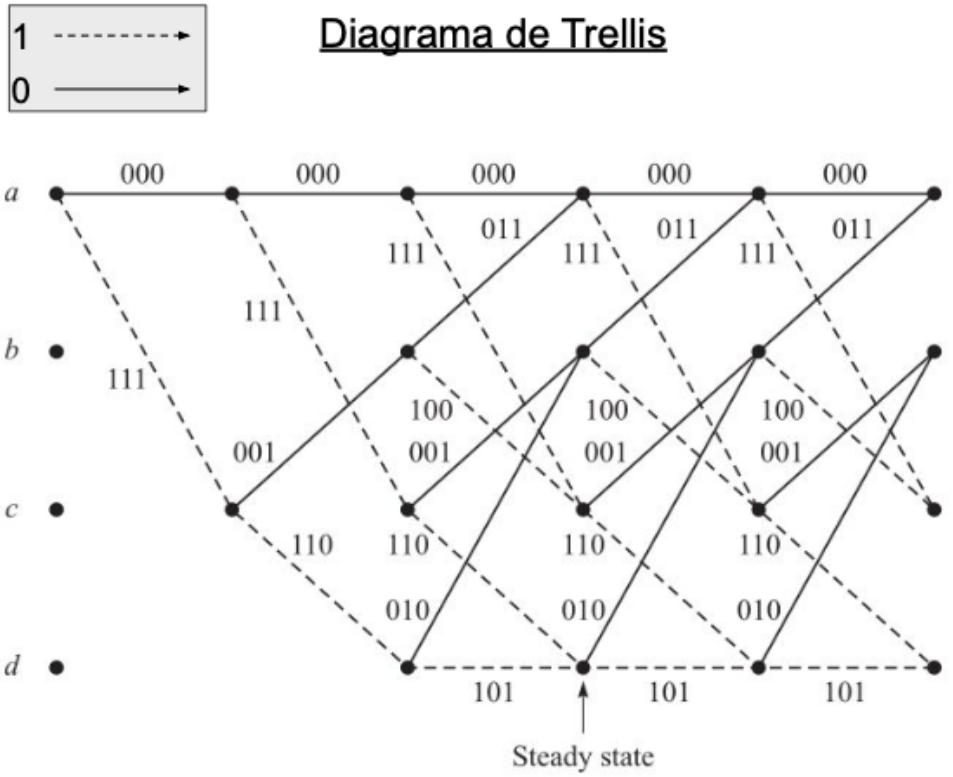
Códigos convolucionales

Algoritmo de Viterbi:

Datos	0	1	0	1	1
TX	000	111	001	100	110
Estado	00	10	01	10	11
	a	c	b	c	d
RX	010	110	011	100	100

¿Cómo se realiza la decodificación?

Con el algoritmo de Viterbi.

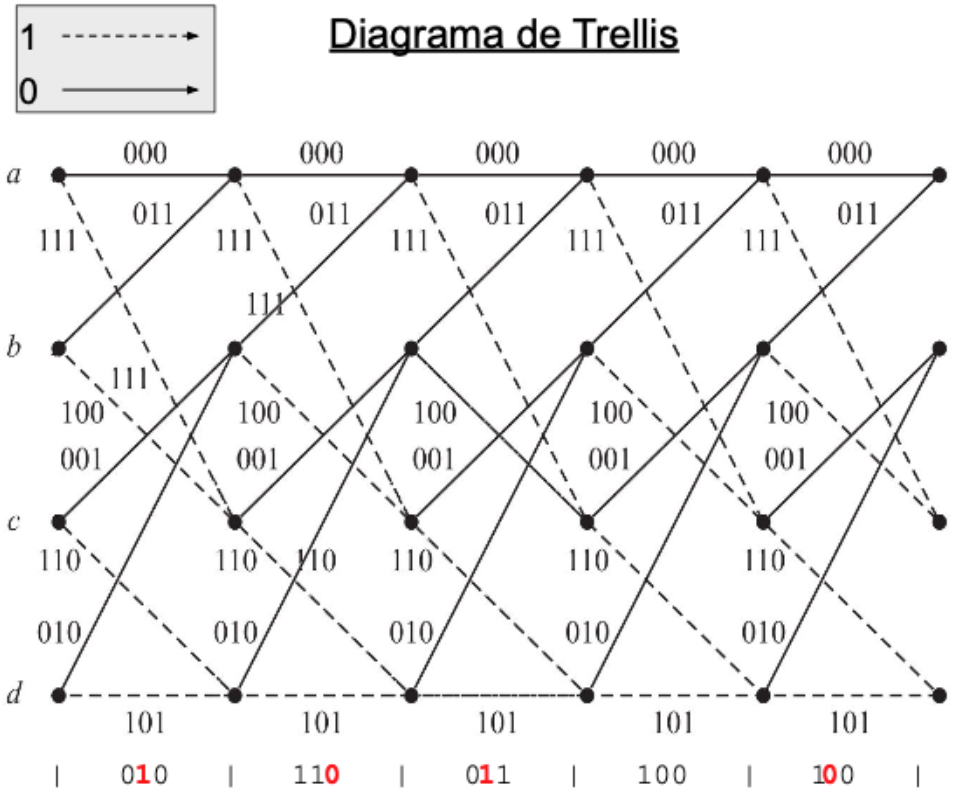


Códigos convolucionales

Algoritmo de Viterbi:

Distancias:

RX	010	110	011	100	100
000	1	2	2	1	1
001	2	3	1	2	2
010	0	1	1	2	2
011	1	2	0	3	3
100	2	1	3	0	0
101	3	2	2	1	1
110	1	0	2	1	1
111	2	1	1	2	2

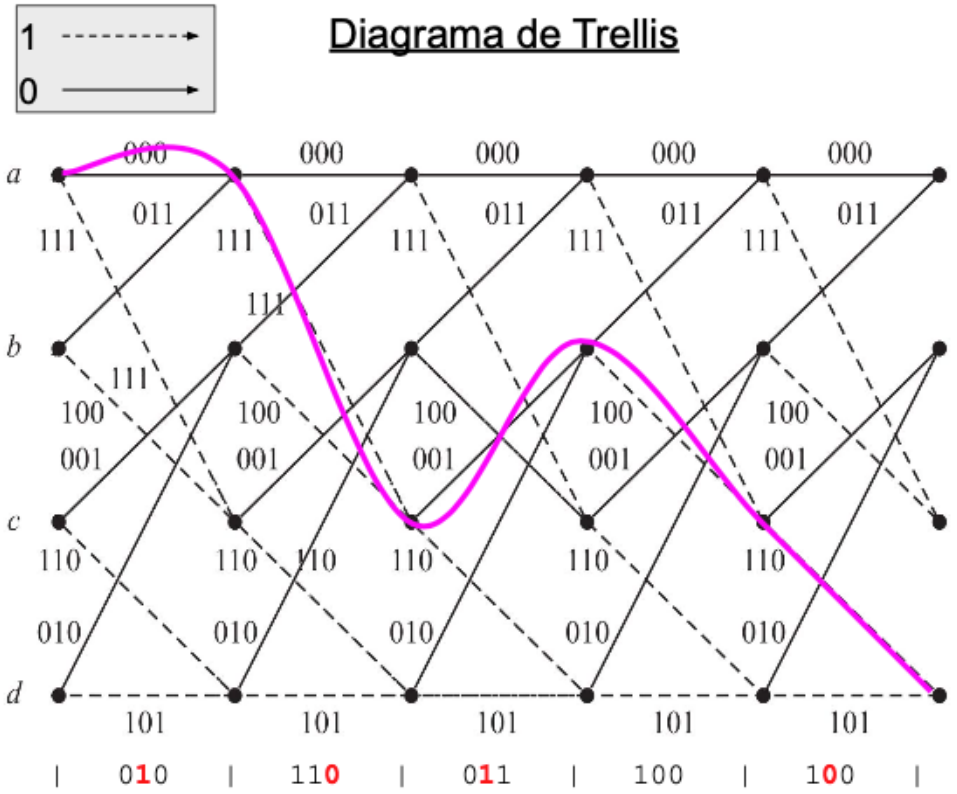


Códigos convolucionales

Algoritmo de Viterbi:

Distancias:

RX	010	110	011	100	100
000	1	2	2	1	1
001	2	3	1	2	2
010	0	1	1	2	2
011	1	2	0	3	3
100	2	1	3	0	0
101	3	2	2	1	1
110	1	0	2	1	1
111	2	1	1	2	2



Códigos convolucionales

- Existen múltiples variantes, dependiendo de los polinomios generadores usados.
- Son excelentes para corregir errores de tipo aleatorio.
- Muy usados en conjunto con códigos Reed-Solomon.
  - Lo cual era la combinación más efectiva hasta hace poco tiempo, y se los denomina **códigos concatenados**.
  - Primero un decodificador interno convolucional, que corrige los errores aleatorios, y luego un decodificador externo RS que corrige errores en ráfagas.

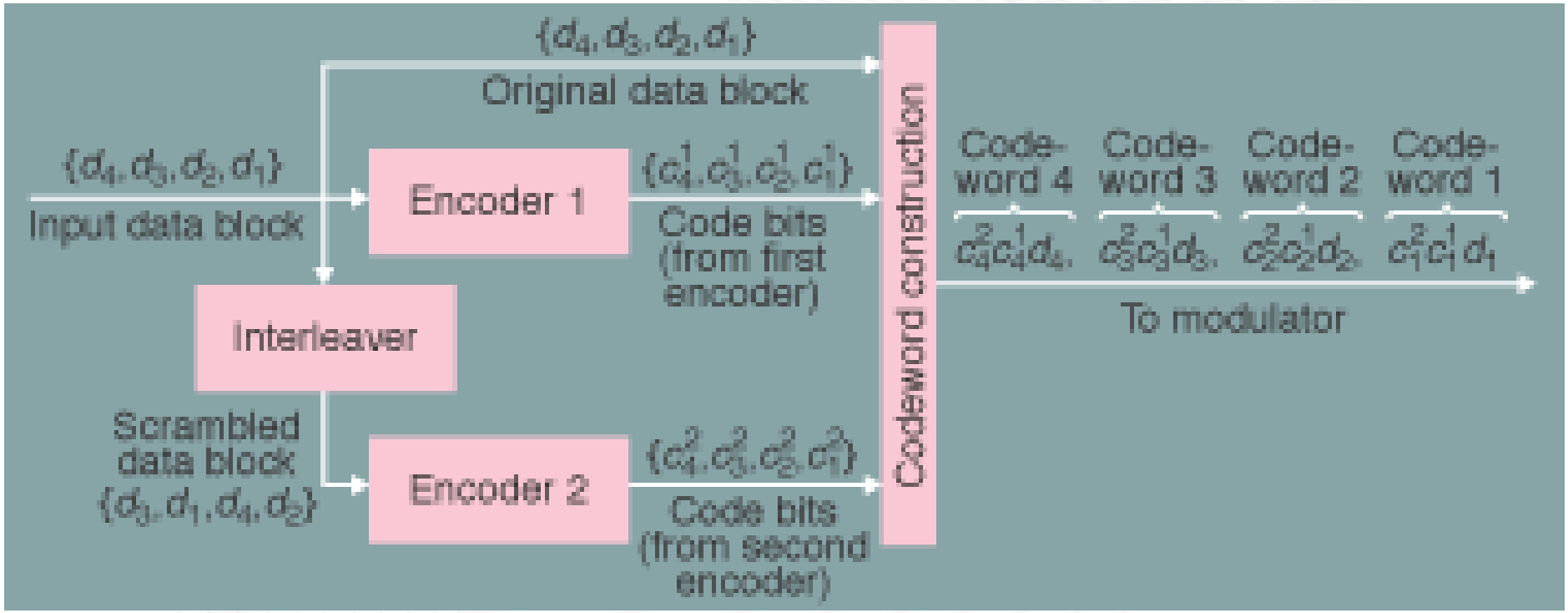
Códigos Turbo (*Turbo codes*)

- Son algunos de los códigos FEC más novedosos.
- Son los primeros en lograr un code rate cercano a la capacidad máxima de Shannon.
  - Generaron una revolución a partir de su aparición, a mediados de los años 90.
- Como muchos de los anteriores, son una familia de códigos, con muchas variantes.
- Usados en comunicaciones móviles 3G y 4G.
- La idea básica es similar a la de los códigos concatenados.
  - Pero en vez de concatenarlos en serie, hacerlo **en paralelo**.

Códigos Turbo (*Turbo codes*)

- La variante más simple, genera tres sub-bloques de bits:
  - El primer sub-bloque, de *k* bits, son los datos originales.
  - El segundo sub-bloque, de *n/2* bits, son bits de paridad calculados sobre los datos originales.
    - Usualmente, se usa algún código convolucional.
  - El tercer sub-bloque, también de *n/2* bits, también son bits de paridad, pero calculados sobre una **permutación conocida** de los datos originales.
    - También calculados con el mismo código convolucional.
  - Estos bloques pueden ser calculados en paralelo.
- A pesar de calcular las paridades mediante códigos convolucionales, los códigos turbo son considerados de bloque, porque los datos de entrada son procesados de esa manera.
- Entonces, el bloque de datos que se transmite tiene *k+n* bits, y el code rate es *k/(k+n)*.

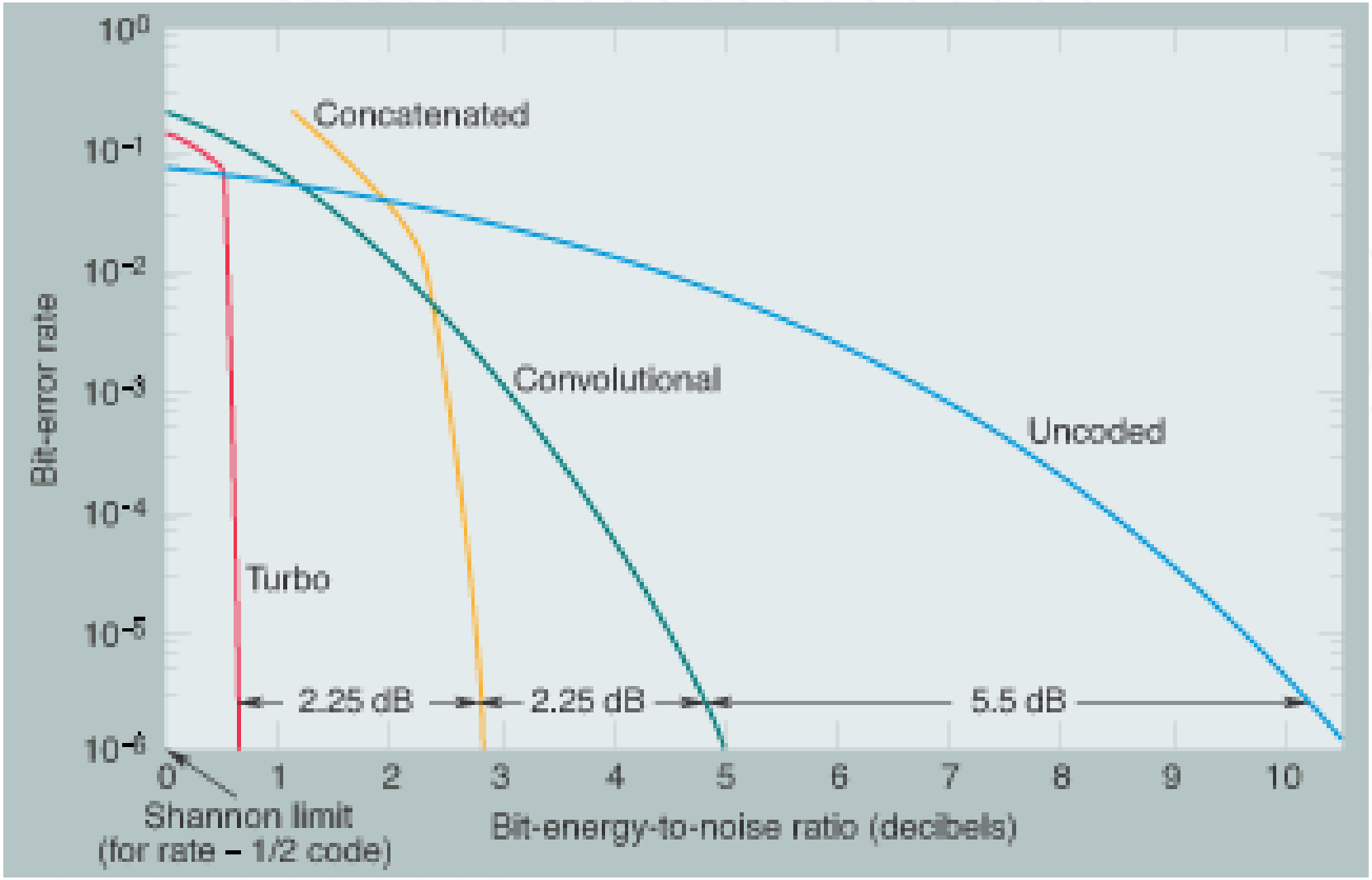
Códigos Turbo (*Turbo codes*)



Códigos Turbo (*Turbo codes*)

- En el decodificador, hay tantos sub decodificadores como sub-bloques de paridad.
- Cada uno de estos sub-decodificadores usa conceptos probabilísticos, similares a los del algoritmo de Viterbi.
  - Pero agregando a cada valor decodificado la probabilidad obtenida.
- Mediante un proceso iterativo, los sub-decodificadores se intercambian los valores obtenidos, hasta converger a una solución.

Comparativa de performance



Resumen

- Objetivo: maximizar la capacidad del canal, sin sacrificar la confiabilidad de los datos.
- Tipos de errores: aleatorios y en ráfagas.
- Distancia mínima de Hamming
  - Determina cuántos errores de bits pueden ser detectados, y con qué probabilidad.
- Detección de errores: agregar información redundante a un mensaje para ayudar a detectar un error.
  - Paridad y CRC como algoritmos de detección más comunes.
  - Si se detecta, normalmente se pide retransmisión.
- Métodos de retransmisión (ARQ): Stop-and-wait, Go-Back-N o Selective Request.
  - Requieren un canal secundario, aumentan la latencia y la congestión, y son inapropiados para ciertas aplicaciones.

Resumen

- Corrección de errores (FEC): busca que el receptor corrija los errores detectados.
  - Dos tipos: de bloques lineales y convolucionales.
  - Capacidad de corrección determinada por su distancia de Hamming mínima.
  - Eficiencia o code rate = k / n; redundancia = n – k.
  - Mayor redundancia → menor eficiencia y menor velocidad de transferencia.
- Códigos de repetición, Códigos de Hamming y Códigos Reed-Solomon.
- Códigos convolucionales, decodificados con el algoritmo de Viterbi.
- Códigos concatenados y códigos Turbo.

Agradecimientos

- Las diapositivas de este tema fueron basadas en las realizadas por el Ing. Esteban Volentini.