

# Introdução a Teoria da Informação

## TI0056 2014.1

### Codificação de Canal

**Prof. Walter C. Freitas Jr**

walter@gtel.ufc.br

Grupo de Pesquisa em Telecomunicações Sem Fio (GTEL)

<http://www.gtel.ufc.br/~walter>

<http://sites.google.com/site/walterjr/>

<http://sites.google.com/site/profwalterfreitas/>

Curso de Graduação em Engenharia de Teleinformática (CGETI)

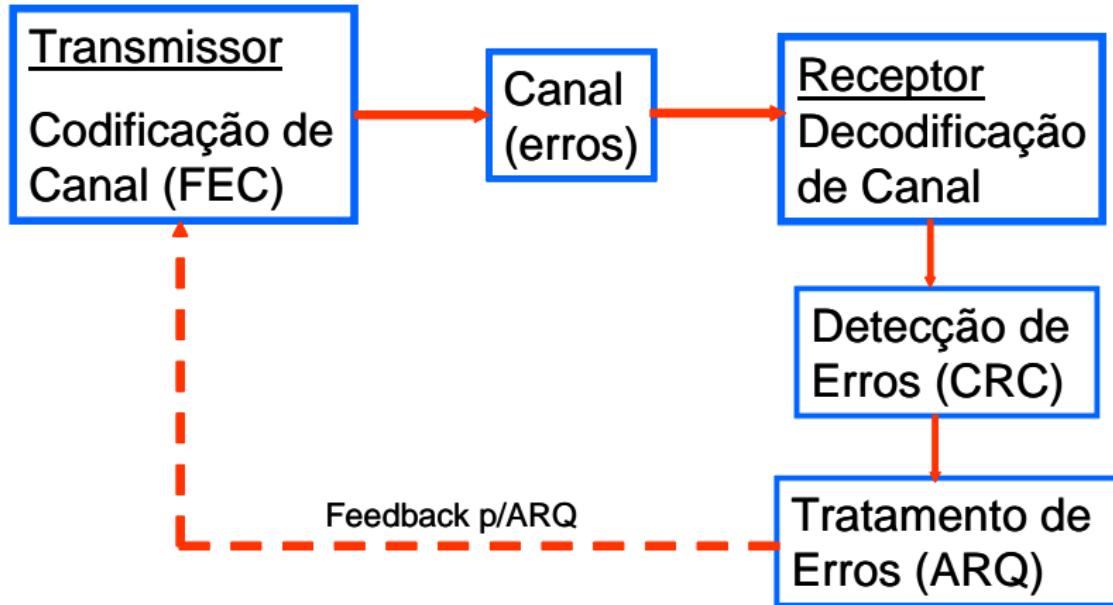
# Introdução

- ▶ Os erros de transmissão em comunicações digitais dependem da relação sinal-ruído ( $S/N$ ).
- ▶ Se a  $S/N$  for fixa e a taxa de erros for muito elevada é preciso melhorar a confiabilidade por outro meio.
- ▶ Uma maneira possível é codificar a informação de modo a detectar ou a corrigir os erros (codificação para controle de erros).

*Códigos para controle de erros:* Envolvem a adição sistemática de dígitos redundantes os quais, só por si, não transportam informação mas tornam possível detectar e até corrigir alguns erros de transmissão.

- ▶ Sonda Voyager I - primeira aplicação bem sucedida de codificação de canal
- ▶ Sistemas de comunicações atuais (e.g. TV digital, comunicações móveis, Banda Larga, etc.) é muito comum encontrar sistemas de codificação para controle de erros
- ▶ Uma outra aplicação é a dos sistemas de armazenamento em disco e em fita magnética, especialmente em equipamento de elevada capacidade e “performance”.
- ▶ Também nas memórias de semicondutores são usadas técnicas de codificação com o intuito é diminuir a taxa de falhas de várias falhas por hora para algumas poucas por ano

# Seqüência de Procedimentos



1. FEC (Forward Error Correction) - adição de redundância
2. Decodificação - correção dos erros por meio da redundância adicionada no transmissor
3. Detecção dos erros - nem todos os erros podem ser corrigidos pelo código
4. Tratamento de erros

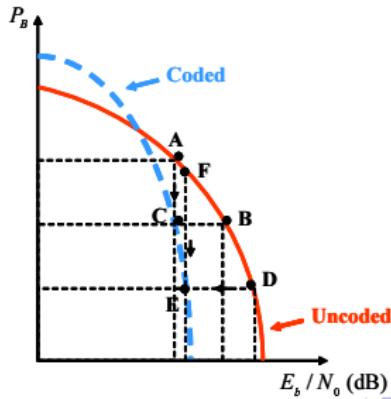
# Códigos de Controle de Erros

## Classificação

- Algébricos: propriedades estruturais para codificar/decodificar - códigos de bloco e códigos cíclicos
- Probabilísticos: códigos convolucionais e turbo

Ganho de Codificação: Para uma dada BER, é a redução na  $E_b/N_0$  que se pode atingir por meio do uso do codificador de canal.

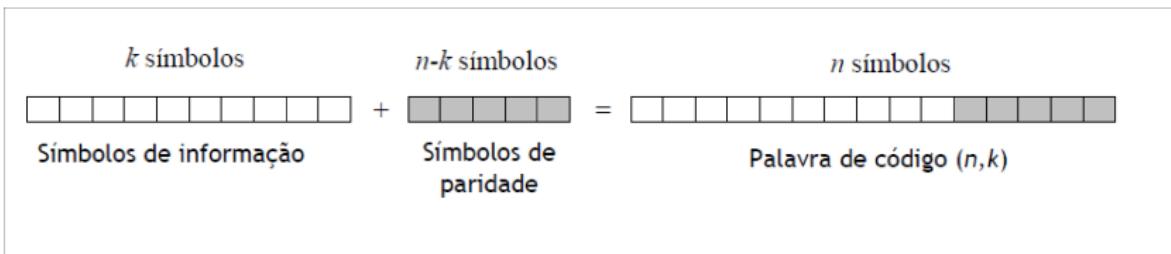
$$G[\text{dB}] = \left( \frac{E_b}{N_0} \right)_u [\text{dB}] - \left( \frac{E_b}{N_0} \right)_c [\text{dB}] \quad (1)$$



# Códigos Corretores de Erros

## Códigos de Bloco - Introdução

- ▶ A fonte binária gera uma seqüência de símbolos à taxa de  $r$  símbolos/s
- ▶ Estes símbolos são agrupados em blocos de  $k$  símbolos
- ▶ A cada um destes blocos de  $k$  símbolos são acrescentados  $n - k$  símbolos redundantes, produzindo uma palavra código de  $n$  símbolos
- ▶ A taxa de símbolos passa a ser  $r_b = r \frac{n}{k}$



# Códigos Corretores de Erros

## Códigos de Bloco - Introdução

**Taxa do Código:**  $R_c = \frac{k}{n}$

**Codificação:** A uma seqüência de informação  $X = (x_1x_2\dots x_k)$  faz-se corresponder uma palavra código  $Y = (y_1y_2\dots y_n)$  de acordo com regras bem definidas

**Decodificação:** A partir da seqüência recebida  $Z = (z_1z_2\dots z_n)$  determina-se a palavra código mais provável, isto é, a palavra de código mais próxima em distância de Hamming da palavra  $Z$



$$x_i, y_i, z_i \in \{0,1\}$$

# Códigos de Bloco

## Distância de Hamming e distância mínima

Seja um vector de código representado por  $X = (x_1 x_2 \dots x_n)$ .

- ▶ Um código é linear quando:
  - Inclui o vector nulo
  - A soma de dois vetores do código é ainda um vector do código
- ▶ Soma de vetores:  $X \oplus Y = (x_1 \oplus y_1 \ x_2 \oplus y_2 \ \dots \ x_n \oplus y_n)$ ,  $x_i, y_i = 0, 1$
- ▶ Peso do vector X:  $w(X)$  número de elementos não nulos do vector
- ▶ Distância de Hamming:  $d(X, Y)$  entre quaisquer dois vetores do código
  - número de elementos diferentes

### Exemplo:

$$X = (1011101) \quad (2)$$

$$Y = (1100101) \quad (3)$$

$$d(X, Y) = 3 \quad (4)$$

$$Z = X \oplus Y = (0111000) \quad (5)$$

$$w(Z) = w(X \oplus Y) = 3 = d(X, Y) \quad (6)$$

# Códigos de Bloco

## Distância de Hamming e distância mínima

- ▶ **Distância mínima de um código:**  $d_{\min}$  é a menor distância de Hamming entre dois vetores válidos do código
- ▶ A distância de Hamming entre  $X$  e  $Y$  é igual ao peso de um outro vector do código,  $X \oplus Y$ . Logo, a menor distância de Hamming é igual ao menor peso. Portanto,

$$d_{\min} = [w(X)]_{\min} \quad X \cong (0000\dots 0) \quad (7)$$

De todos os vetores não nulos um apresenta o menor peso. Esse peso é a distância mínima do código

# Códigos de Bloco

## Capacidade de Correção e Detecção

- ▶ A detecção de erros é sempre possível quando o número de erros de transmissão numa palavra de código é inferior a distância mínima  $d_{min} \Rightarrow$  a palavra errônea não é um vector válido
- ▶ Inversamente, se o número de erros iguala ou excede  $d_{min}$  a palavra errônea pode corresponder a outro vector válido e os erros não podem ser detectados
- ▶ Se um código detecta até  $l$  erros por palavra:  $l = d_{min} - 1$
- ▶ Se um código corrige até  $t$  erros por palavra:  $t = \lfloor \frac{d_{min}-1}{2} \rfloor$
- ▶ Se um código corrige até  $t$  erros por palavra e detecta  $l > t$  erros por palavra:  $d_{min} = t + l + 1$
- ▶ A distância mínima de um código de blocos  $(n, k)$  é limitada superiormente por

$$d_{min} \leq n - k + 1 \tag{8}$$

# Códigos de Bloco

## Exemplos - Capacidade de Correção e Detecção

- $d_{min} = 5$



Este código consegue detectar até 4 erros por palavra. Consegue corrigir até 2 erros por palavra.

- Código de repetição tripla:

$$0 \rightarrow 000$$

$$1 \rightarrow 111 \quad (\text{só tem estas duas palavras de código})$$

$d_{min} = 3 \Rightarrow$  Pode detectar  $l \leq 3 - 1 = 2$  erros/palavra de 3 bits

Pode corrigir  $t \leq \frac{3-1}{2} = 1$  erro/palavra de 3 bits

- $d_{min} = 7$ , é possível:

- Corrigir erros triplos ( $t = 3$ )
- Corrigir erros duplos ( $t = 2$ ) e detectar erros quádruplos ( $l = 4$ )

# Códigos de Bloco

Vale a pena codificar?

- ▶ Seja,  $S$  a potência do sinal,  $T_w$  a duração de uma palavra de  $k$  símbolos antes da codificação (= duração da palavra de  $n$  símbolos depois da codificação) e a energia/palavra de código  $E_s = ST_w$
- ▶ Energia recebida/símbolo (sem codificação)  $E_b = \frac{ST_w}{k}$
- ▶ Energia recebida/símbolo (com codificação)  $E_{bc} = \frac{ST_w}{n} = \frac{k}{n}E_b$
- ▶ Como  $n > k$ , a energia por símbolo diminui com o uso de codificação  
⇒ a probabilidade de erro num símbolo é maior com codificação do que sem codificação
- ▶ No entanto a redundância introduzida pelos  $n - k$  símbolos de paridade permite corrigir erros, o que pode conduzir a uma melhoria global do desempenho do sistema
- ▶ Uma medida da eficiência da codificação obtém-se comparando a probabilidade de erro numa palavra codificada,  $P_{enc}$ , com a probabilidade de erro numa palavra não codificada,  $P_e$

# Códigos de Bloco

Vale a pena codificar?



Seja

- ▶  $p = Q\left(\sqrt{\frac{2E_b}{N_0}}\right)$  - probabilidade de erro em um símbolo, sem codificação
- ▶  $p_c = Q\left(\sqrt{2\frac{k}{n}\frac{E_b}{N_0}}\right)$  - probabilidade de erro num símbolo, com codificação
- ▶ **Sem codificação:**
  - ▶ Probabilidade de erro em uma palavra é igual a 1 menos a probabilidade de todos os  $k$  símbolos da palavra serem recebidos corretamente:

$$P_e = 1 - P(0, k) = 1 - (1 - p)^k \quad (9)$$

- ▶ **Com codificação:**

- ▶ Probabilidade de erro em uma palavra código é a probabilidade de haver  $t + 1$  erros,  $t + 2$  erros,  $t + 3$  erros, etc

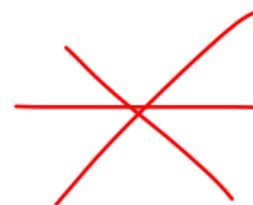
$$P_{enc} = \sum_{i=t+1}^n P(i, n) = \sum_{i=t+1}^n \binom{n}{i} p_c^i (1 - p_c)^{n-i} \quad (10)$$

# Códigos de Bloco

## Exemplo

- Código (23,12) e modulação BPSK

**Sem codificação:**



$$p = Q\left(\sqrt{\frac{2E_b}{N_0}}\right) \quad (11)$$

$$P_e = 1 - (1 - p)^{12} \quad (12)$$

**Com codificação:**

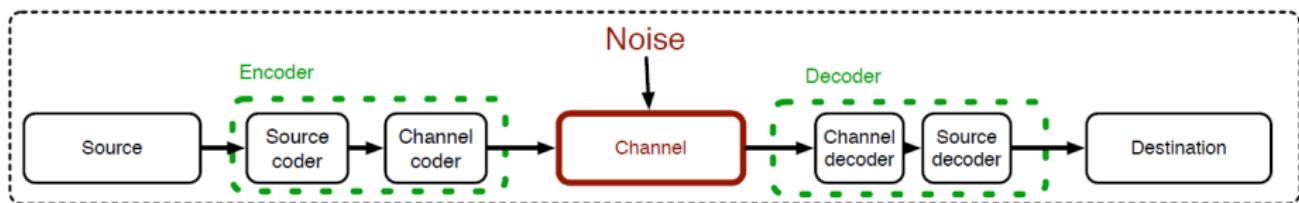
$$p_c = Q\left(\sqrt{2\frac{12}{23}\frac{E_b}{N_0}}\right) \quad (13)$$

$$P_{enc} = \sum_{i=4}^{23} \binom{23}{i} p_c^i (1 - p_c)^{23-i} (t = 3) \quad (14)$$

# Codificação de canal - BSC

## Exemplo Ilustrativo - Transmissão de Imagens

- ▶ Imagens – lena e xadrez
- ▶ Luminância – valores dos pixels [0-255]
- ▶ Canal BSC – probabilidade de erro  $p$
- ▶ *Source code*: comprimento fixo  $\lceil \log_2 256 \rceil$
- ▶ Código de repetição: Taxa:  $R_c = 1/n$



# Codificação de canal - BSC

Exemplo Ilustrativo - Lena

Imagen Transmitida



Imagen Recibida  $p = 0.1$



# Codificação de canal - BSC

Exemplo Ilustrativo - Lena

Não Codificada



$n = 2, BER = 0.1001$



# Codificação de canal - BSC

Exemplo Ilustrativo - Lena

Não Codificada



$n = 4, BER = 0.0279$



# Codificação de canal - BSC

Exemplo Ilustrativo - Lena

Não Codificada



$n = 8, BER = 0.0028$



# Codificação de canal - BSC

Exemplo Ilustrativo - Lena

Não Codificada



$$n = 10, \text{BER} = 9.0504e - 004$$



# Representação Matricial dos Códigos de Blocos

## Códigos Sistemáticos

- ▶ Os primeiros  $k$  símbolos da palavra de código  $\mathbf{Y}$  de  $n$  bits constituem a sequência de informação  $\mathbf{X} = (x_1 x_2 \dots x_k)$ , e os últimos  $n - k$  símbolos representam os bits de verificação ou bits de paridade (ou o contrário)

$$\mathbf{Y} = (x_1 x_2 \dots x_k c_1 c_2 \dots c_{n-k}) = (\mathbf{X} | \mathbf{C}) \quad (15)$$

- ▶  $\mathbf{X}$  – Vetor de mensagem
- ▶  $\mathbf{C}$  – Vetor de paridade (redundância)
- ▶ Cada palavra de código  $\mathbf{Y}$  é obtida multiplicando o vetor  $\mathbf{X}$  por uma matriz  $\mathbf{G}$ , chamada de matriz geradora, de dimensão  $k \times n$

$$\mathbf{Y} = \mathbf{X}\mathbf{G} \quad (16)$$

# Representação Matricial dos Códigos de Blocos

## Matriz geradora

- A matriz geradora de um código sistemático tem a seguinte estrutura:

$$\mathbf{G} = \left[ \begin{array}{ccccccc|cc} 1 & 0 & 0 & \cdots & 0 & g_{1,k+1} & \cdots & g_{1,n} \\ 0 & 1 & 0 & & 0 & g_{2,k+1} & \cdots & g_{2,n} \\ 0 & 0 & 1 & \cdots & 0 & \vdots & & \vdots \\ \vdots & \vdots & \vdots & \cdots & \vdots & \vdots & & \vdots \\ 0 & 0 & 0 & \cdots & 1 & g_{k,k+1} & \cdots & g_{k,n} \end{array} \right] = \left[ \mathbf{I}_k \mid \mathbf{P} \right]$$

$\underbrace{\hspace{10em}}$   $k \times k$        $\underbrace{\hspace{10em}}$   $k \times (n-k)$

$\mathbf{I}_k$  — Matriz identidade  $k \times k$

$\mathbf{P}$  — Submatriz  $k \times (n-k)$

# Representação Matricial dos Códigos de Blocos

## Matriz geradora

**Determinação da matriz  $C$ :** ► Como  $\mathbf{Y} = \mathbf{X}\mathbf{G} = (\mathbf{X}|\mathbf{C})$  e  $\mathbf{G} = (\mathbf{I}_k|\mathbf{P}) \Rightarrow \mathbf{C} = \mathbf{X}\mathbf{P}$

- A questão está em determinar a submatriz  $\mathbf{P}$  para obtermos os valores pretendidos de  $d_{min}$  e  $R_c$

**Matriz de verificação de paridade  $H$ :** ► Relacionada com  $\mathbf{G}$  temos a matriz de verificação de paridade,  $\mathbf{H}$ , de dimensões  $n \times (n - k)$ , tal que  $\mathbf{GH} = \mathbf{0}$ :

$$\mathbf{H} = \begin{bmatrix} \mathbf{P} \\ \mathbf{I}_{n-k} \end{bmatrix} = \begin{bmatrix} g_{1,k+1} & \cdots & g_{1,n} \\ g_{2,k+1} & \cdots & g_{2,n} \\ \vdots & & \\ g_{k,k+1} & \cdots & g_{k,n} \\ 1 & \cdots & 0 \\ \cdots & \cdots & \cdots \\ 0 & \cdots & 1 \end{bmatrix}$$

- O produto de qualquer palavra código pela matriz  $\mathbf{H}$  é um vetor nulo:  $\mathbf{YH} = \mathbf{XGH} = \mathbf{0}$

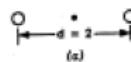
# Exemplo de Código de Blocos

## Código de Hamming

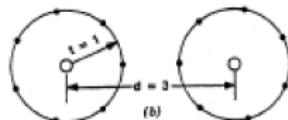
- ▶ Um código de Hamming é um código linear de blocos  $(2^{n-k} - 1, k)$  com
  - ▶  $n - k \geq 3$  bits de paridade
  - ▶  $n = 2^{n-k} - 1$
  - ▶ Independentemente do número de bits de paridade a distância mínima é sempre  $d_{\min} = 3$
  - ▶ Um código de Hamming é um código que permite corrigir 1 erro ou detectar 2 erros.
  - ▶ As  $k$  linhas da submatriz **P** consistem em todas as palavras de  $n - k$  bits com dois ou mais 1's (devido a **H**), em qualquer ordem.

# Distância Mínima e Capacidade de Correcção

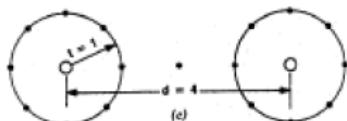
Exemplos com códigos binários



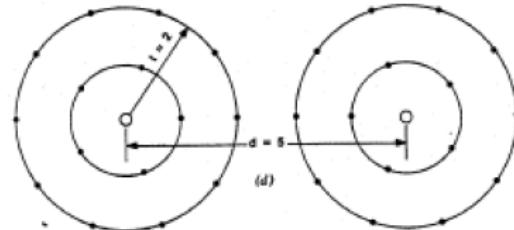
$(n, n-1)$ , Paridade par



$(7, 4)$ , Hamming



$(8, 4)$ , Hamming aumentado



$(5, 1)$ , código de repetição

# Exemplo de Codificação

## FAQ

P.: Como calcular as palavras de um código linear?

R.: Como a soma de duas palavras de um código linear é uma palavra do mesmo código, podemos obter palavras à custa de outras já determinadas.

P.: Quais?

R.: Havendo  $k$  bits de informação só precisamos usar  $k$  palavras código linearmente independentes, isto é, um conjunto de  $k$  palavras nenhuma das quais pode ser obtida por combinação linear de 2 ou mais palavras do conjunto. As restantes  $2^k - k$  palavras são obtidas das primeiras por adição módulo 2. Isto quer dizer que não precisamos de calcular  $2^k$  palavras recorrendo à matriz geradora: basta calcular  $k$ .

P.: Como determinar  $k$  palavras linearmente independentes?

R.: Uma maneira fácil é escolher aquelas que só têm um 1 nas primeiras  $k$  posições, isto é, as mensagens de  $k$  bits com peso 1. As operações anteriores equivalem a somar linhas da matriz geradora (ver exemplo seguinte).

# Codificação Linear

## FAQ

- ▶ Consideremos o código  $(7, 4)$  com a matriz geradora seguinte:

$$G = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 & 1 \end{bmatrix}$$

- ▶ Existem  $2^k = 2^4 = 16$  palavras de código correspondentes a outras tantas mensagens de  $k = 4$  bits. Nestas 16 mensagens há 4 com peso unitário. As palavras de código correspondentes são:

Mensagens de 4 bits	Palavras de código com 7 bits
(1) 1000	1000101
(2) 0100	0100111
(3) 0010	0010110
(4) 0001	0001011

- ▶ As restantes 12 palavras podem ser obtidas a partir destas. Por exemplo, qual é a palavra de código correspondente à mensagem  $\mathbf{X} = [0101]$ ? Basta adicionar a 2da e a 4ta palavras:  $0001011 \oplus 0100111 = 0101100$
- ▶ A operação anterior é equivalente à soma das linhas de ordem 2 e 4 da matriz geradora  $\mathbf{G}$ :  $\mathbf{Y} = \mathbf{XG} = [0101]\mathbf{G} = [0101100]$

# Como determinar a distância mínima a partir da matriz H?

- A matriz de verificação de paridade pode ser escrita linha-a-linha como

$$H = \begin{bmatrix} h_1 \\ \vdots \\ h_i \\ \vdots \\ h_{n-k} \end{bmatrix} \quad (h_i \text{ — vector-linha de ordem } i)$$

- Ora já sabemos que se  $\mathbf{Y}$  for um vetor de código então  $\mathbf{YH} = \mathbf{0}$ . Suponhamos então que o vetor  $\mathbf{Y}$  tem elementos não-nulos nas posições  $i, j$  e  $k$ , por exemplo. Será fácil de verificar que as linhas  $h_i, h_j$  e  $h_k$  somadas dão zero
- Se houver uma palavra de código de peso  $L$  então existirão  $L$  linhas de  $\mathbf{H}$  que somadas dão zero
- Como a distância mínima do código é igual ao peso mínimo de todas as palavras de código, então existem pelo menos  $d_{\min}$  linhas de  $\mathbf{H}$  que somadas dão zero
- O menor número de linhas de  $\mathbf{H}$  cuja soma é nula é igual à distância mínima do código

# Códigos de Bloco

## Decodificação

- ▶ Força bruta - comparar a palavra recebida com todas as possíveis  $2^k$  palavras do código
- ▶ Código de Hamming com  $R_c \geq 0,8$
- ▶  $n - k \geq 5 \Rightarrow n \geq 31, k \geq 26$
- ▶ Seria preciso guardar  $n \times 2^k > 10^9$  bits para comparação!!
- ▶ Há outros métodos mais práticos, associados à matriz de verificação de paridade **H**
- ▶ Como  $\mathbf{ZH} = (00\dots 0)$ , se **Z** pertencer ao conjunto das palavras código, e se **Z** não pertencer então  $\mathbf{ZH} \neq (00\dots 0) \Rightarrow$  pelo menos um elemento será não nulo
- ▶ A decodificação faz-se multiplicando a sequência recebida  $\mathbf{Z} = (z_1 z_2 \dots z_n)$  pela matriz de verificação de paridade **H**, o que dá um vetor de  $n - k$  bits:  $\mathbf{S} = \mathbf{ZH}$ , **S** é a síndrome
  - ▶ Uma síndrome não nula indica a presença de erros
  - ▶ Uma síndrome nula significa que ou não houve erros introduzidos na transmissão, ou houve erros na transmissão que transformaram a palavra código enviada numa outra palavra código válida (erros não detectados)
  - ▶ Se o código tiver uma distância mínima  $d_{\min}$  são precisos pelo menos  $d_{\min}$  erros para transformar uma palavra código em outra palavra código igualmente válida

# Códigos de Bloco

## Decodificação com Síndrome

- ▶ A sequência recebida  $\mathbf{Z}$  é a soma em módulo 2 da palavra de código  $\mathbf{Y}$  com um eventual vetor binário de erro,  $\mathbf{E}$ ,  $\mathbf{S} = \mathbf{ZH} = (\mathbf{Y} \oplus \mathbf{E})\mathbf{H} = \mathbf{YH} \oplus \mathbf{EH} = \mathbf{EH}$
- ▶ Sendo  $\mathbf{Z} = \mathbf{Y} \oplus \mathbf{E}$ , então,  $\mathbf{Y} = \mathbf{Z} \oplus \mathbf{E}$
- ▶ O vetor de erro,  $\mathbf{E}$ , tem  $n$  bits  $\Rightarrow$  existem  $2^n$  padrões de erro possíveis
- ▶ A síndrome,  $\mathbf{S}$ , tem  $n - k$  bits  $\Rightarrow$  existem  $2^{n-k} < 2^n$  síndromes possíveis
- ▶ A síndrome não determina univocamente  $\mathbf{E}$
- ▶ Dos  $2^n$  padrões possíveis de erro apenas podem ser corrigidos  $2^{n-k} - 1$  padrões (exclui-se o padrão nulo)
- ▶ É conveniente que os  $2^{n-k} - 1$  padrões que podem ser corrigidos sejam os padrões de erro mais prováveis, isto é, aqueles que apresentem menos erros (por outras palavras, aqueles que tenham um peso mais baixo)

# Decodificação com Síndromes

## Máxima verossimilhança

1. A partir dos  $2^{n-k} - 1$  padrões de erro mais prováveis calcula-se uma tabela de  $2^{n-k} - 1$  síndromes  $S = EH$  possíveis.
2. Tendo recebido uma palavra  $Z$  de  $n$  bits calcula-se a síndrome respectiva,  $S = ZH$ .
3. Consulta-se a tabela de síndromes para determinar o padrão de erro mais provável,  $\hat{E}$ , que corresponde à síndrome calculada.
4. A palavra transmitida mais provável,  $\hat{Y}$ , obtém-se adicionando a palavra recebida,  $Z$ , ao padrão de erro estimado.

# Decodificação com Síndromes

## Máxima verossimilhança

- Se  $\mathbf{E} = [e_1 e_2 \dots e_j \dots e_n]$  e  $\mathbf{H} = [h_1 h_2 \dots h_j \dots h_n]^T$ , ( $h_j$  vetor linha de  $n - k$  símbolos)

$$\mathbf{S} = \mathbf{EH} = [e_1 e_2 \dots e_j \dots e_n] \cdot [h_1 h_2 \dots h_j \dots h_n]^T \quad (17)$$

- Imaginemos que ocorreu um único erro, isto é,  $\mathbf{E}$  é nulo exceto na posição  $j$

$$\mathbf{E} = [00\dots 1\dots 0] \Rightarrow \mathbf{S} = [00\dots 1\dots 0], \mathbf{H} = h_j \quad (18)$$

- Isto significa que, num código corretor de erros simples como o de Hamming, se a síndrome for igual à linha de ordem  $j$  da matriz  $\mathbf{H}$  então houve um erro no  $i$ -ésimo bit da palavra código recebida
- Para permitir a detecção de erros, as linhas da matriz de verificação de paridade devem ser distintas (para que não haja ambiguidade) e diferentes de zero
  - Número de linhas de  $\mathbf{H}$  -  $n$
  - Número de linhas com  $n - k$  bits, distintas e diferentes de zero,  $2^{n-k} - 1$
  - $2^{n-k} - 1$  (código de Hamming, precisamente)

# Decodificação com Síndromes

## Exemplo

- Consideremos o código de Hamming (7, 4) anterior e seja  $\mathbf{G}$  a sua matriz geradora:

$$\mathbf{G} = [\mathbf{I}_k \mid \mathbf{P}] = [\mathbf{I}_4 \mid \mathbf{P}] = \left[ \begin{array}{cccc|ccc} 1 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 & 1 \end{array} \right]$$

- A matriz de verificação de paridade vale

$$\mathbf{H} = \left[ \begin{array}{c} \mathbf{P} \\ \mathbf{I}_{n-k} \end{array} \right] = \left[ \begin{array}{c} \mathbf{P} \\ \mathbf{I}_3 \end{array} \right] = \left[ \begin{array}{ccc} 1 & 0 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 0 \\ 0 & 1 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{array} \right]$$

# Decodificação com Síndromes

## Exemplo

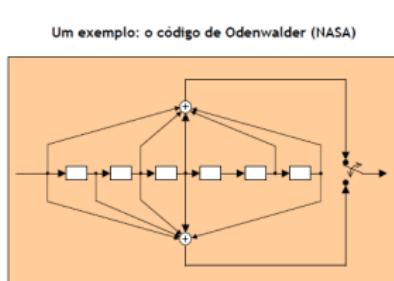
- As síndromes são as linhas de  $\mathbf{H}$  e há  $2^{n-k} - 1 = 7$  padrões de erro corrigíveis, que são os 7 vetores de erro com menor peso:

S	$\hat{\mathbf{E}}$
0 0 0	0 0 0 0 0 0 0
1 0 1	1 0 0 0 0 0 0
1 1 1	0 1 0 0 0 0 0
1 1 0	0 0 1 0 0 0 0
0 1 1	0 0 0 1 0 0 0
1 0 0	0 0 0 0 1 0 0
0 1 0	0 0 0 0 0 1 0
0 0 1	0 0 0 0 0 0 1

- Este código  $(7, 4)$  permite detectar erros duplos (pois  $d_{\min} = 3$ ) mas apenas corrigir erros simples.

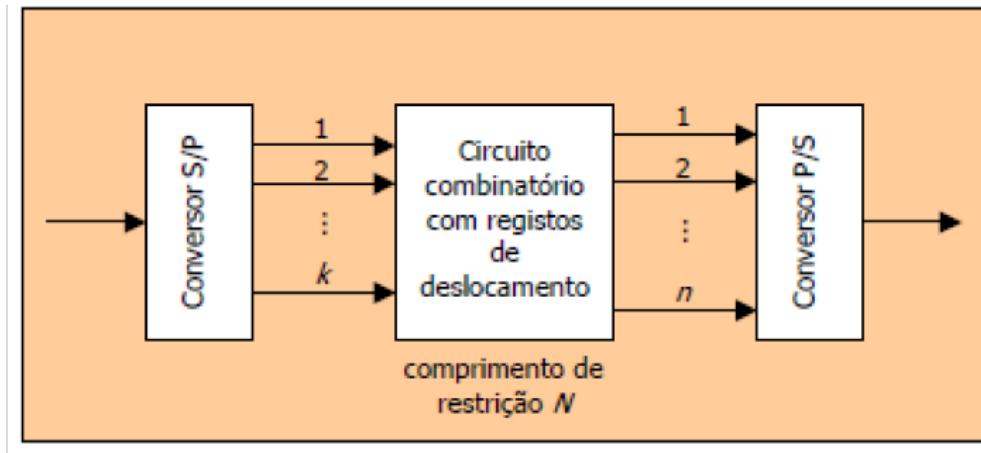
# Códigos Convolucionais

- ▶ O processamento não se faz em bloco com palavras código, como nos códigos algébricos (de blocos)
- ▶ São códigos em árvore, representáveis por diagramas de estados
- ▶ Usam-se registradores de deslocamento na codificação (memória)
- ▶ O “hardware” de codificação convolucional é mais simples que o “hardware” de codificação por blocos (por exemplo, não é preciso “buffer” de entrada)
- ▶ A estrutura convolucional é adequada a comunicações espaciais e por satélite:
  - ▶ requer codificadores simples (no “espaço”)
  - ▶ a elevada ”performance” do código obtém-se com métodos de descodificação sofisticados (em “Terra” )



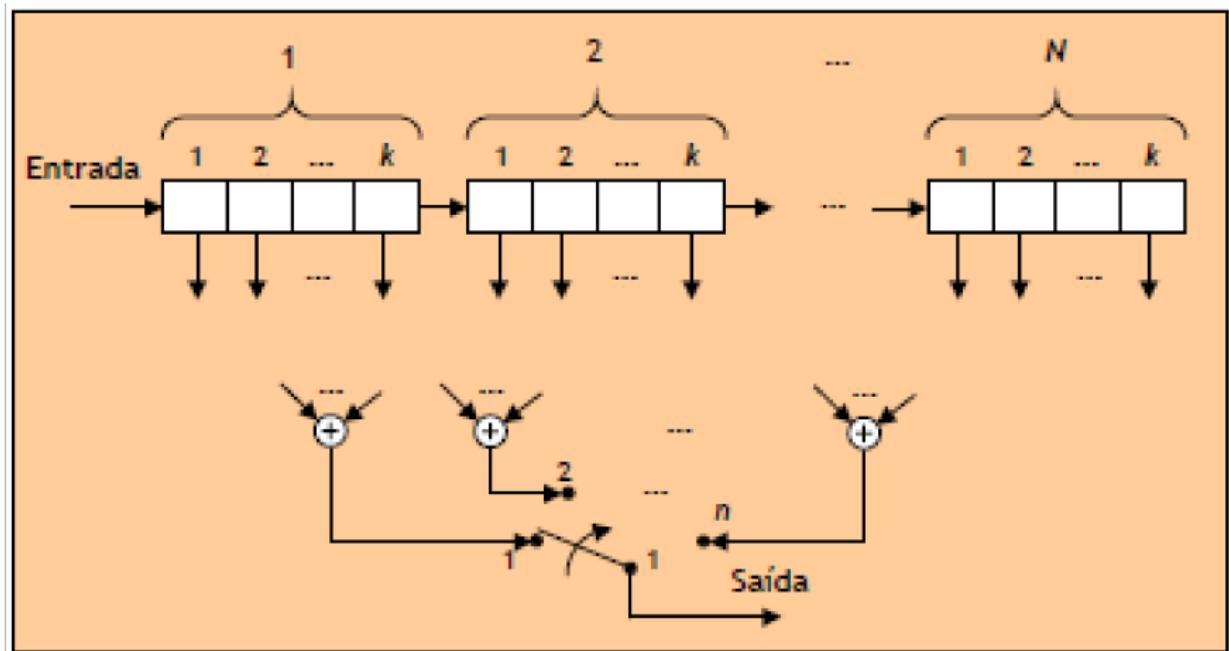
# Códigos Convolucionais

Diagrama de blocos genérico de um codificador convolucional  $(n, k)$



# Códigos Convolucionais

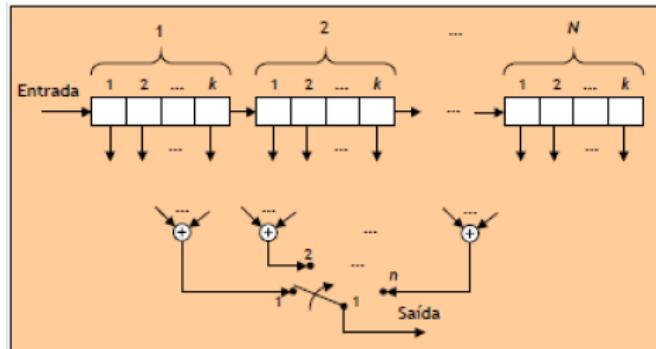
Diagrama de blocos genérico de um codificador convolucional  $(n, k)$



# Códigos Convolucionais

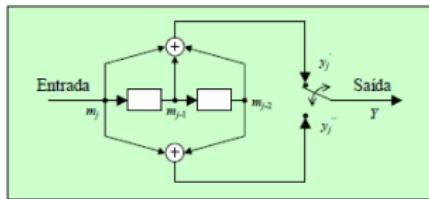
## Parâmetros importantes

- ▶ Taxa do código:  $R_c = k/n$
- ▶ Comprimento de restrição:  $N = \max_{1 \leq i \leq k} N_i$
- ▶ Memória do codificador:  $v = \sum_{i=1}^k (N_i - 1)$ , se  $N_i$  são iguais  
 $v = k(N_i - 1)$
- ▶ Número de estados:  $2^v$ , se  $N_i$  são iguais  $2^v = 2^{k(N_i - 1)}$



# Códigos Convolucionais

## Exemplo codificador convolucional (2, 1, 3)



- ▶ Bits de entrada:  $k = 1$
- ▶ Bits de saída:  $n = 2$
- ▶ Bits de estado:  $v = N - 1 = 2$
- ▶ Este codificador gera  $n = 2$  bits,  $y'_j$  e  $y''_j$ , por cada bit de entrada:

$$y'_j = m_j \oplus m_{j-1} \oplus m_{j-2} \quad y''_j = m_j \oplus m_{j+2} \quad (19)$$

- ▶ A sequência binária de saída é  $Y = y'_1 y''_1 y'_2 y''_2 y'_3 y''_3 y'_4 y''_4$
- ▶ Memória do codificador:  $v = N - 1 = 2$
- ▶ Existem  $2^{k(N-1)} = 4$  estados diferentes: 00, 01, 10 e 11
- ▶ Saídas e estados (supondo que inicialmente o registo está limpo  $m_0 m_{-1} = 00$ )

- ▶ entrada  $m_1 = 0 \Rightarrow$  saída  $y'_1 y''_1 = 00$
- ▶ entrada  $m_1 = 1 \Rightarrow$  saída  $y'_1 y''_1 = 11$
- ▶ Taxa do código:  $R_c = k/n = 1/2$
- ▶ Um bit de entrada influencia  $nN = 6$  bits sucessivos de saída.
- ▶ Normalmente  $n$  e  $k$  são pequenos e o comprimento de restrição toma valores inferiores a 10

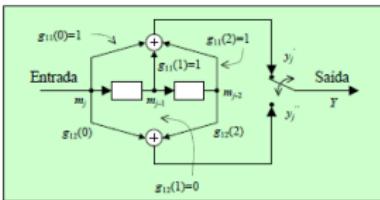
# Códigos Convolucionais

## Representações

- ▶ Métodos de representação gráfica
  - ▶ Árvore do código
  - ▶ Treliça ("trellis") do código - método "condensado"
  - ▶ Diagrama de estados - método "condensado"
- ▶ Métodos de representação não gráfica
  - ▶ Vetores de ligação (ou vetores geradores)
  - ▶ Polinômios de ligação (ou polinômios geradores)
  - ▶ Matriz geradora
  - ▶ Resposta impulsional
- ▶ Métodos de decodificação:
  - ▶ Máxima verossimilhança (Algoritmo de Viterbi) - método algorítmico ("software")
  - ▶ Decodificação sequencial (Wozencraft-Fano) - método algorítmico ("software")
  - ▶ Descodificação com "feedback"
- ▶ O algoritmo de Viterbi requer "hardware" complexo para cálculo e armazenamento de informação
- ▶ A decodificação sequencial é de complexidade média. A "performance" é próxima do método anterior em certos casos
- ▶ A decodificação com "feedback" é o método mais simples, mas de todos o menos confiável

# Códigos Convolucionais

## Exemplo de Representação



- ▶ Parâmetros do código:  $n = 2$ ,  $k = 1$  e  $N = 3$
- ▶ Vetores de ligação:  $g_1 = [111]$  e  $g_2 = [101]$
- ▶ Resposta impulsional:

Conteúdo do registo	Palavra	
	$y'_j$	$y''_j$
1 0 0	1	1
0 1 0	1	0
0 0 1	1	1

Sequência de entrada:

1

0

0

Sequência de saída:

11

10

11  $\leftarrow$  Resposta impulsional do codificador

Seja a sequência de entrada  $m = 1\ 0\ 1$ . A saída pode ser determinada pela *sobreposição ou adição linear de "impulsos"* de entrada deslocados no tempo (isto é, é a convolução da sequência de entrada com a resposta impulsional).

Entrada	Saída				
1	11	10	11		
0		00	00	00	
1		11	10	10	11
Soma (mód. 2)	11	10	00	10	11

# Códigos Convolucionais

## Exemplo de Representação

- ▶ Polinômios de ligação
  - ▶ O codificador pode ser representado por  $n$  polinômios geradores binários de grau  $N - 1$  ou menor
  - ▶ No exemplo apresentado:  $g_1(D) = 1 + D + D^2$  e  $g_2(D) = 1 + D^2$
- ▶  $D$  é um operador de atraso unitário
- ▶ A sequência de saída,  $Y(D)$ , vem dada por  $Y(D) = m(D)g_1(D)$  entrelaçado com  $m(D)g_2(D)$
- ▶ Exemplo: codificador convolucional (GSM)
- ▶ Header & Data:

$$G4 = 1 + D^2 + D^3 + D^5 + D^6 \quad (20)$$

$$G7 = 1 + D + D^2 + D^3 + D^6 \quad (21)$$

$$G5 = 1 + D + D^4 + D^6 \quad (22)$$

# Códigos Convolucionais

## Exemplo de Representação

- Se a sequência de entrada for  $m = 101$ , então,  $m(D) = 1 + D^2$ :

$$m(D)g_1(D) = (1+D^2)(1+D+D^2) = 1+D+D^3+D^4$$

$$m(D)g_2(D) = (1+D^2)(1+D^2) = 1+D^4$$

↓

$$m(D)g_1(D) = 1+D+0D^2+D^3+D^4$$

$$m(D)g_2(D) = 1+0D+0D^2+0D^3+D^4$$

↓

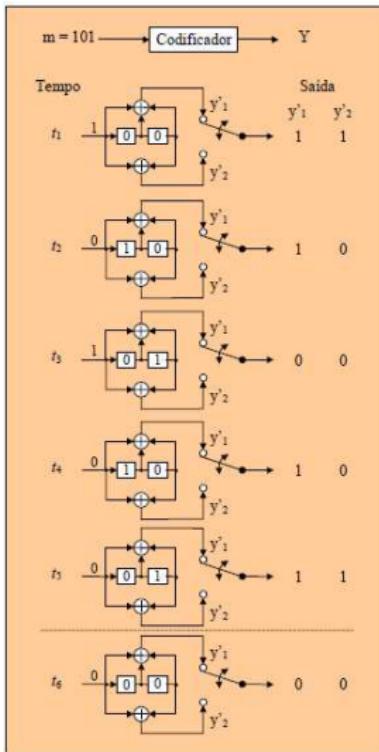
$$Y(D) = (1,1) + (1,0)D + (0,0)D^2 + (1,0)D^3 + (1,1)D^4$$

↓

$$Y = 11 \ 10 \ 00 \ 10 \ 11$$

# Códigos Convolucionais

## Exemplo de Funcionamento



► Sequência de saída:  $Y = 1110001011$

# Códigos Convolucionais

## Representações não gráficas

- Os vetores ou os polinômios geradores dão origem à matriz geradora, de dimensões  $k \times n$

$$\mathbf{G} = \begin{bmatrix} \mathbf{g}_{11} & \mathbf{g}_{12} & \cdots & \mathbf{g}_{1,n} \\ \vdots & & \ddots & \\ \mathbf{g}_{k,1} & \mathbf{g}_{k,2} & \cdots & \mathbf{g}_{k,n} \end{bmatrix}$$

$$\mathbf{G}(D) = \begin{bmatrix} g_{11}(D) & g_{12}(D) & \cdots & g_{1,n}(D) \\ \vdots & & \ddots & \\ g_{k,1}(D) & g_{k,2}(D) & \cdots & g_{k,n}(D) \end{bmatrix}$$

# Códigos Convolucionais

## Resumo das Representações não gráficas

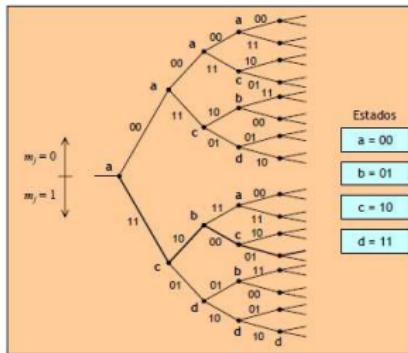
### Resumo de representações da matriz geradora

- vectores binários:  $G = \begin{bmatrix} 010 & 111 & 111 \\ 101 & 011 & 100 \end{bmatrix}$
- vectores em octal:  $G = \begin{bmatrix} 2 & 7 & 7 \\ 5 & 3 & 4 \end{bmatrix}$
- polinómios binários:  $G(D) = \begin{bmatrix} D & 1+D+D^2 & 1+D+D^2 \\ 1+D^2 & D+D^2 & 1 \end{bmatrix}$

# Códigos Convolucionais

Representações gráficas dos códigos convolucionais: árvore

- ▶ Árvore do código  $(2, 1, 3)$  apresentado antes



- ▶ Há  $2^j$  ramos possíveis para o  $j$ -ésimo bit de mensagem
- ▶ O padrão de ramos começa a repetir-se em  $j = 3$  porque é  $N = 3$
- ▶ Como há repetição poderemos usar duas outras formas de representação: **a treliça e o diagrama de estados**

# Códigos Convolucionais

Representações gráficas dos códigos convolucionais: a treliça e o diagrama de estados

Treliça

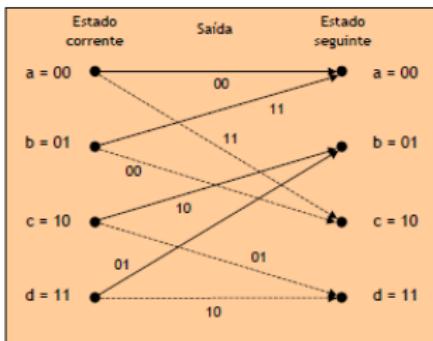
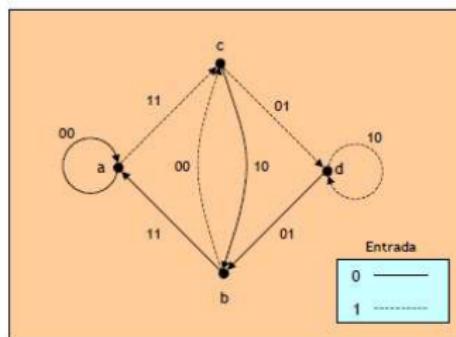


Diagrama de estados



# Códigos Convolucionais Punctionados (*Punctured*)

## Punctionamento

- ▶ Imaginemos que temos um código de taxa  $1/2$  e que, em cada dois pares de bits de saída, eliminamos (isto é, não transmitimos) o último bit. Deste modo, dois bits de entrada vão dar origem não a quatro bits de saída, como seria normal, mas apenas a três tendo o código resultante código punctionado (perfurado) uma taxa de  $2/3$ .
- ▶ Os códigos perfurados são muito usados dada a sua flexibilidade pois
  - ▶ Torna-se possível não só obter códigos com diversas taxas de código a partir de um único codificador-pai mas também fazer a decodificação de todos eles com o mesmo descodificador
  - ▶ A decodificação torna-se mais simples relativamente aos códigos não-perfurados com a mesma taxa
- ▶ É possível alterar a taxa do código durante a transmissão conforme as características do canal: enquanto houver pouco ruído usa-se uma taxa elevada ( $5/6$ , por exemplo) mas se o canal se tornar mais ruidoso muda-se para uma taxa mais baixa ( $2/3$  ou  $1/2$ , por exemplo), pois tem uma distância livre maior ⇒ **Redundância Incremental (IR)**
- ▶ Em qualquer dessas situações o decodificador é sempre o mesmo, o que é uma vantagem

# Códigos Convolucionais Punctionados (*Punctured*)

Exemplo com os códigos perfurados das normas DVB-S e DVB-T

- ▶ Código-pai: taxa 1/2, comprimento de restrição 7, polinômios geradores 171 e 133

Taxas do código	Padrão de perfuração	Sequência transmitida (após conversão paralelo-série)	Distância livre
1/2	$y':1$ $y':1$	$y'_1 y'_1$ (1-2)	10
2/3	$y':10$ $y':11$	$y'_1 y'_2 y'_2$ (1-2-X-4)	6
3/4	$y':101$ $y':110$	$y'_1 y'_1 y'_2 y'_3$ (1-2-X-4-5-X)	5
5/6	$y':10101$ $y':11010$	$y'_1 y'_1 y'_2 y'_3 y'_4 y'_5$ (1-2-X-4-5-X-X-8-9-X)	4
7/8	$y':1000101$ $y':1111010$	$y'_1 y'_1 y'_2 y'_3 y'_4 y'_5 y'_6 y'_7$ (1-2-X-4-X-6-X-8-9-X-X-12-13-X)	3

- ▶ Se a sequência de saída do código de taxa 1/2 for 111011001001 a sequência de saída do código perfurado de taxa 3/4 será 11010000:
  - ▶ Saída do codificador de taxa 1/2: 11 10 11 00 10 01
  - ▶ Punctionamento ou perfuração: 11 X0 1X 00 X0 0X
  - ▶ Saída do codificador de taxa 3/4: 1 1 0 1 0 0 0 0

# Códigos Convolucionais

## Ganho de codificação

- ▶ Para obtermos uma determinada BER num sistema de comunicações sem codificação necessitamos de uma determinada relação  $E_b/N_0$ . Com codificação adequada necessitaremos de uma menor relação  $E_b/N_0$ . A diferença, em dB, entre esses dois valores representa o ganho de codificação do código. O ganho de codificação máximo (ganho de codificação assimptótico) atinge-se quando a relação  $E_b/N_0$  é muito elevada
- ▶ *Hard-Decision* (Decisão Abrupta): Ganho de codificação  $\leq 10 \log \frac{R_{cdf}}{2}$
- ▶ *Soft-Decision* (Decisão Suave): Ganho de codificação  $\leq 10 \log(R_{cdf})$
- ▶ De uma maneira geral, com decisões suaves o ganho de codificação é cerca de 3 dB maior

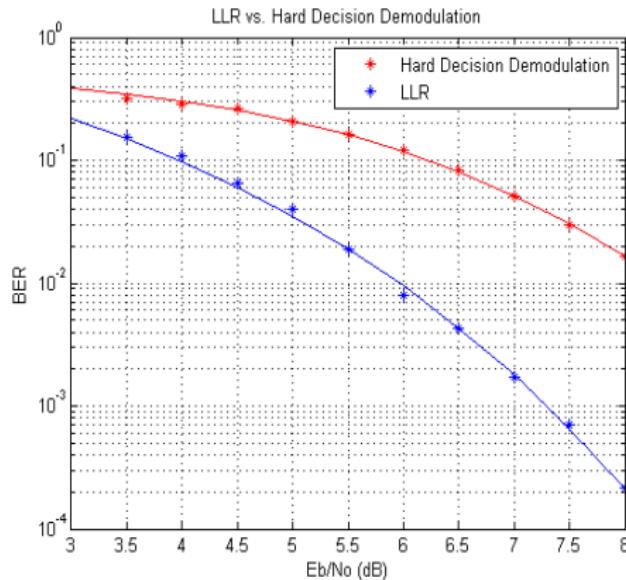
Ganho de codificação assimptótico,  $G_a$ , com decisões rígidas

Comprimento de restrição, N	Taxa 1/2		Taxa 1/3	
	Distância livre	Ganho de codificação máximo (dB)	Distância livre	Ganho de codificação máximo (dB)
3	5	0,97	8	1,25
4	6	1,76	10	2,22
5	7	2,43	12	3,01
6	8	3,01	13	3,36
7	10	3,98	15	3,98
8	10	3,98	16	4,26
9	12	4,77	18	4,77
10	12	4,77	20	5,23

# Códigos Convolucionais

Ganho de codificação: *Hard* vs. *Soft*

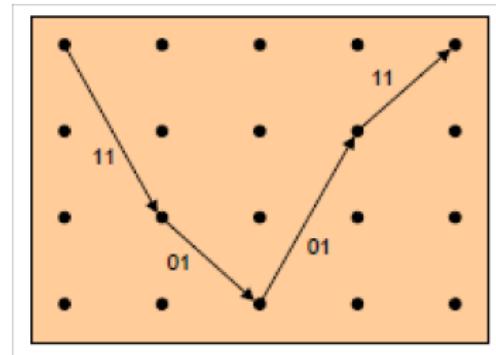
- ▶ Código convolucional  $R=1/2$
- ▶ Canal AWGN
- ▶ Modulação 16-QAM
- ▶ Algoritmo de Viterbi: *Hard-decision* vs. *Soft-decision* (LLR)



# Distâncias euclidianas e distâncias de Hamming

## Exemplo

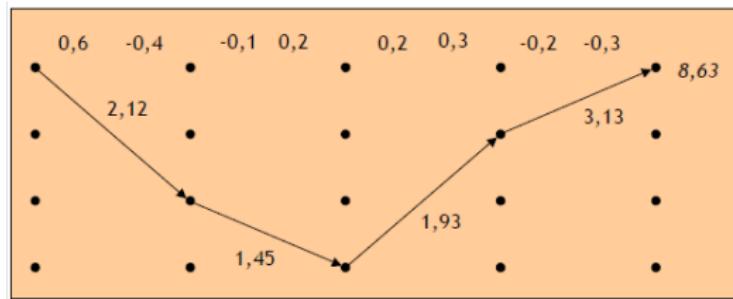
- ▶ *Soft-Decision*: o decodificador recebe valores reais  $\{-\infty \leftrightarrow +\infty\}$
- ▶ *Hard-Decision*: o decodificação recebe valores binários  $\{0, 1\}$
- ▶ Seja demodulação suave, recebendo-se a sequência de reais  $z_l$   
0, 6; 0, 4; 0, 1; 0, 2; 0, 2; 0, 3; 0, 2; 0, 3. Qual é a sua distância euclidiana quadrática ao percurso da figura seguinte?



# Distâncias euclidianas e distâncias de Hamming

## Exemplo

- ▶ Os bits 0 e 1 da figura deverão ser convertidos em  $y_l = \pm 1$  (fazendo  $0 \rightarrow -1$  e  $1 \rightarrow +1$ ). Assim, o primeiro ramo, 11, corresponde ao ponto de coordenadas  $(+1, +1)$ ; o segundo ramo corresponde ao ponto de coordenadas  $(-1, +1)$
- ▶ A distância euclidiana de cada ramo é:  $\sum_{j=1}^n (z_{lj} - y_{lj}^{(i)})^2$  (se  $n = 2$ ):  
 $[z_{l1} - y_{l1}^{(i)}]^2 + [z_{l2} - y_{l2}^{(i)}]^2$
- ▶ Exemplo de distância euclidiana do par  $(-0,1, 0,2)$  ao ponto  $(-1, +1)$  (segundo ramo):  $(-0,1 + 1)^2 + (0,2 - 1)^2 = 1,45$
- ▶ Métrica acumulada:  $2,12 + 1,45 + 1,93 + 3,13 = 8,63$



# Decodificador de máxima verossimilhança

## Algoritmo de Viterbi

- ▶ Suponhamos que recebemos a sequência  $Z = Y + E$ . Na treliça ou na árvore o percurso de  $Z$  diverge do percurso de  $Y \rightarrow$  pode ou não ser um percurso válido
- ▶ Se não for um percurso válido um decodificador de máxima verossimilhança (*Maximum Likelihood* - ML) tenta encontrar o percurso válido mais provável
- ▶ Mas, há  $2^n$  percursos possíveis para uma sequência-mensagem arbitrária de  $n$  bits!
- ▶ No algoritmo de Viterbi limita-se a comparação a  $2^{k(N-1)}$  percursos sobreviventes (que é um numero independente do tamanho da mensagem,  $n$ )
- ▶ Isto torna viável a decodificação de máxima verossimilhança, baseada em métricas e percursos sobreviventes

# Decodificador de máxima verossimilhança

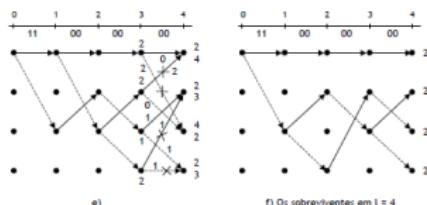
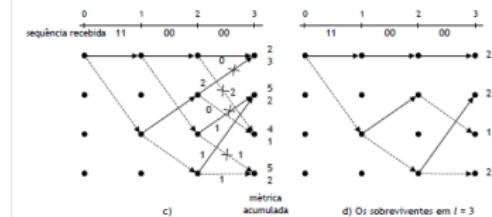
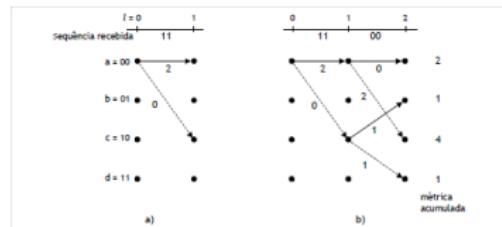
## Algoritmo de Viterbi

- ▶ O algoritmo de Viterbi atribui uma métrica a cada ramo de cada percurso sobrevivente
- ▶ A métrica é igual à distância de Hamming do ramo correspondente de  $Z$ , se se tratar de demodulação *hard* (supõe-se  $P_0 = P_1 = 1/2$ )
- ▶ A métrica é igual à distância euclidiana, se se tratar de demodulação *soft*
- ▶ A métrica de um percurso é igual à soma das métricas dos ramos constituintes
- ▶ O decodificador deve calcular 2 métricas por cada nó e armazenar  $2^{k(N-1)}$  percursos sobreviventes, cada um com  $n$  ramos
- ▶ Sempre que dois percursos convergem num nó, sobrevive aquele que tiver menor métrica. Em caso de empate escolhe-se um à sorte
- ▶  $Z$  é decodificado (estimado) como o percurso sobrevivente com menor métrica

# Decodificador de máxima verossimilhança

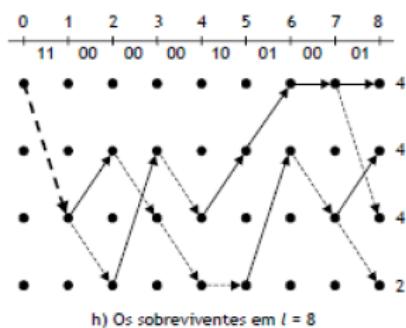
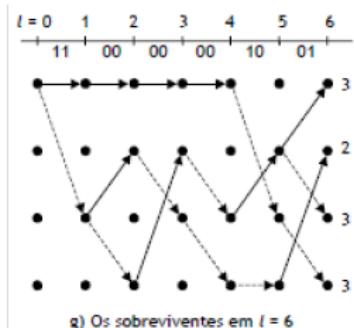
## Exemplo - Algoritmo de Viterbi

- Mensagem: 1 0 1 1 1 0 1 1 0 0
- Sequência codificada: 11 10 00 01 10 01 00 01 01 11
- Sequência recebida: 11 00 00 00 10 01 00 01 01 11

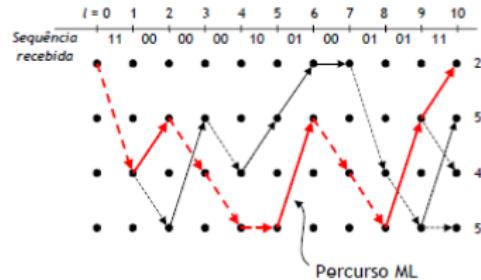


# Decodificador de máxima verossimilhança

Exemplo - Algoritmo de Viterbi



O percurso de máxima verosimilhança (percurso  $ML$ )

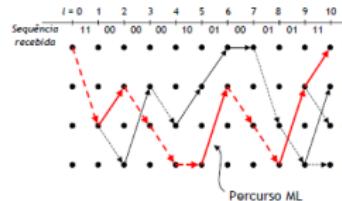


# Decodificador de máxima verossimilhança

## Exemplo - Algoritmo de Viterbi

- ▶ Mensagem enviada: 1 0 1 1 1 0 11 0 0
- ▶ Sequência estimada: 1110000110010001 0111
- ▶ Mensagem estimada: 1 0 1 1 1 0 1 1 0 0
- ▶ Portanto, o algoritmo corrigiu os dois erros que sabíamos existirem na sequência recebida

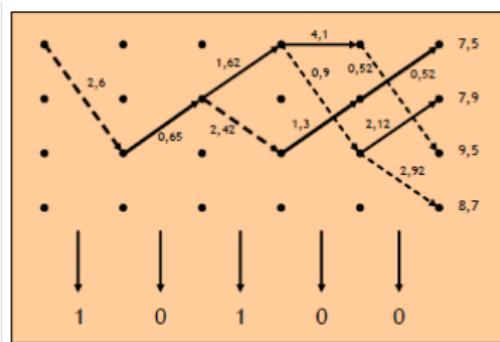
O percurso de máxima verosimilhança (percurso *ML*)



# Decodificador de máxima verossimilhança

## Exemplo *Soft-Decision* - Algoritmo de Viterbi

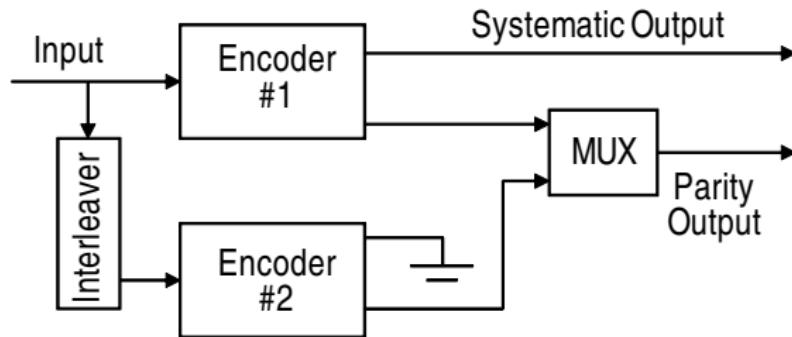
- ▶ Codificador (2, 1, 3) habitual
- ▶ Mensagem: 10100
- ▶ Sequência codificada: 11 10 00 10 11
- ▶ Sequência recebida: -0,6; 0,8; 0,3; -0,6; 0,1; 0,1; 0,7; 0,1; 0,6; 0,40
- ▶ O percurso ML é aquele cuja distância euclidiana quadrática à sequência recebida é a menor
- ▶ Exemplo de distância euclidiana quadrática: 1ro ramo:  
$$(1 + 0,6)^2 + (1 - 0,8)^2 = 2,6$$
- ▶ O percurso ML tem uma métrica total de 7,5:



# Códigos Turbo (*Turbo Codes*)

## Introdução - Códigos Concatenados

- ▶ Serial - *outer e inner codes* p/ melhorar o desempenho
- ▶ Paralelo - Ao invés da tradicional concatenação em serial códigos podem também serem conectados em paralelo
- ▶ Turbo Codes: Concatenação de dois códigos convolucionais sistemáticos recursivos (RSC) em paralelo



# Códigos Turbo (*Turbo Codes*)

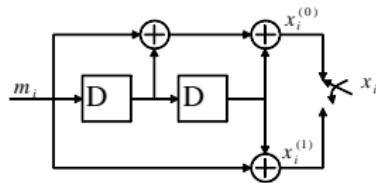
## *Interleaving*

- ▶ Dilema
  - ▶ Shannon mostrou que para códigos aleatórios com comprimento de bloco “suficientemente grande” poderia se atingir a capacidade de canal com probabilidade de erro próxima de zero
  - ▶ Códigos:
    - ▶ decodificação vs. Complexidade
    - ▶ Estrutura vs. aleatórios
    - ▶ *“Almost all codes are good, except those that we can think of.”*
- ▶ Solução:
  - ▶ Fazer o código “parecer” aleatório, enquanto é mantida uma estrutura suficiente para permitir: decodificação vs. Complexidade:  
Entrelaçamento tem esse propósito
  - ▶ Turbo codes + entrelaçamento: propriedade de códigos aleatórios
  - ▶ Padrão de entrelaçamento conhecido no Rx: decodificação possível

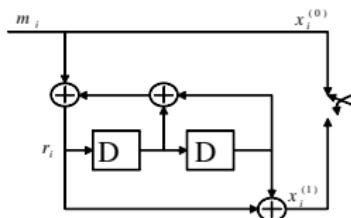
# Códigos Turbo (*Turbo Codes*)

## Recursive Systematic Convolutional (RSC)

- ▶ Codificadores RSC podem ser construídos de códigos convolucionais tradicionais realimentando uma das possíveis saídas do código
- ▶ Nesse caso a resposta impulso do código é infinita
- ▶ Uma entrada arbitrária causará uma “boa” (peso de Hamming) saída com uma alta probabilidade. Algumas entradas causarão uma “má” saída



Constraint Length K= 3



# Códigos Turbo (*Turbo Codes*)

## Justificativas - RSC+*Interleaving*

- ▶ Em sistemas codificados o desempenho é ditado pelas palavras código de baixo peso
- ▶ Bom código - produz palavra-código de baixo peso com baixa probabilidade
- ▶ RSC: palavra-código de baixo peso, mas eventualmente ainda produz palavra-código de baixo peso para alguns vetores de entrada
- ▶ Razão do entrelaçamento - a probabilidade de que ambos codificadores tenham entrada que causem palavras código de baixo peso é diminuída
- ▶ Concatenação paralela + entrelaçamento  $\Rightarrow$  “bom” código

# Códigos Turbo (*Turbo Codes*)

Princípio TURBO 0.5dB de Shannon

- ▶ Princípio turbo conceito mais geral do que apenas considerar a decodificação turbo
- ▶ Princípio turbo: *“Never discard information prematurely that may be useful in making a decision until all decisions related to that information have been completed.”* - Andrew Viterbi
- ▶ *“It is a capital mistake to theorize before you have all the evidence. It biases the judgment.”* -Sir Arthur Conan Doyle
- ▶ Tal princípio pode ser usado para interfacear sistemas que empreguem múltiplos algoritmos baseados em treliça: Códigos concatenados e/ou Codificação+equalização

# Códigos Turbo (*Turbo Codes*)

## Limite de Shannon

- ▶ Capacidade de canal: máxima taxa de transmissão de dados no qual um comunicação livre de erros é possível
- ▶ Capacidade de canal AWGN (Teorema da capacidade de Shannon-Hartley):

$$C = W \log_2 \left( 1 + \frac{S}{N} \right) \quad (23)$$

- ▶  $W$  [Hz] - largura de banda
- ▶  $S = E_b C$  [W] - pot. média do sinal recebido
- ▶  $N = N_0 W$  - pot. média do ruído

# Códigos Turbo (*Turbo Codes*)

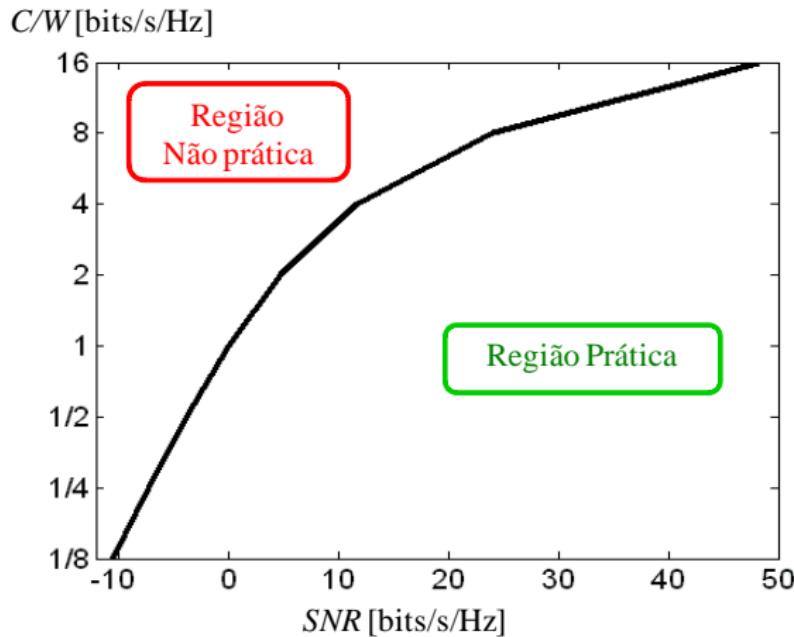
## Limite de Shannon

- ▶ O teorema de Shannon estipula um limite na taxa de transmissão de dados, não a probabilidade de erro:
  - ▶ O teorema diz que é possível se transmitir informação a uma taxa  $R_b$ , no qual  $R_b \leq C$  com uma probabilidade de erro arbitrariamente pequena usando um esquema de codificação apropriado (*code length*)
  - ▶ Para uma dada taxa  $R_b > C$ , não é possível encontrar um código que possa atingir uma probabilidade de erro arbitrariamente pequena

$$C = W \log_2 \left( 1 + \frac{S}{N} \right) \quad (24)$$

# Códigos Turbo (*Turbo Codes*)

Limite de Shannon



# Códigos Turbo (*Turbo Codes*)

## Limite de Shannon

$$\frac{C}{W} = \log_2 \left( 1 + \frac{E_b}{N_0} \frac{C}{W} \right) \quad (25)$$

- ▶ Existe um valor limite para a Eb/No no qual existe uma comunicação sem erros p/ qualquer taxa de informação
- ▶ Aumentando-se a largura de banda isoladamente, a capacidade não pode ser aumentada para qualquer valor desejável

# Implicações do teorema da capacidade de informação

- ▶ Sendo o canal limitado em faixa e potência, qual o impacto do teorema da capacidade de informação?
- ▶ *Framework* prático - **sistema ideal**
- ▶ Taxa de bits  $R_b$  igual à capacidade de informação  $C$
- ▶ Potência média de transmissão

$$P = E_b C \quad (26)$$

em que  $E_b$  é a **energia por bit**

- ▶ Logo, para o sistema ideal

$$\frac{C}{B} = \log \left( 1 + \frac{E_b}{N_0} \frac{C}{B} \right) \quad (27)$$

# Implicações do teorema da capacidade de informação

Um sistema ideal com largura de banda infinita tem uma capacidade de canal finita:

$$C = B \log_2 \left( 1 + \frac{S}{N_0 B} \right) \quad (28)$$

$$C = \frac{B}{\ln 2} \ln \left( 1 + \frac{S}{N_0 B} \right) \quad (29)$$

Se  $B \rightarrow \infty$   $\frac{S}{N_0 B} \rightarrow 0$ . Portanto

$$C_\infty = \lim_{B \rightarrow \infty} \approx \frac{B}{\ln 2} \frac{S}{N_0 B} = \frac{1}{\ln 2} \frac{S}{N_0} \quad (30)$$

$$\boxed{C_\infty = 1,44 \frac{S}{N_0}} \quad (31)$$

# Implicações do teorema da capacidade de informação

Existe um valor limite de  $E_b/N_0$  (limite de Shannon) abaixo do qual não pode haver comunicação sem erros, qualquer que seja o ritmo de transmissão. Sendo  $T$  a duração de cada bit,  $E_b$  a sua energia e  $R = 1/T$  bits/s, então

$$\frac{S}{N} = \frac{E_b/T}{N_0 B} = \frac{E_b}{N_0} \frac{R}{B} \Rightarrow C = B \log_2 \left( 1 + \frac{S}{N} \right) = B \log_2 \left( 1 + \frac{E_b}{N_0} \frac{R}{B} \right) \quad (32)$$

Mas  $R \leq C \Rightarrow \frac{R}{B} \leq \frac{C}{B} = \left( 1 + \frac{E_b}{N_0} \frac{R}{B} \right)$

Se  $B \rightarrow \infty \Rightarrow \frac{R}{B} \leq \log_2 \left( 1 + \frac{E_b}{N_0} \frac{R}{B} \right) = \frac{\ln \left( 1 + \frac{E_b}{N_0} \frac{R}{B} \right)}{\ln 2} \approx \frac{\frac{E_b}{N_0} \frac{R}{B}}{\ln 2}$

Portanto,

$$\frac{E_b}{N_0} \geq \ln 2 = 0.693 \text{ } (-1.59 \text{dB})$$

Limite de Shannon

(33)

# Implicações do teorema da capacidade de informação

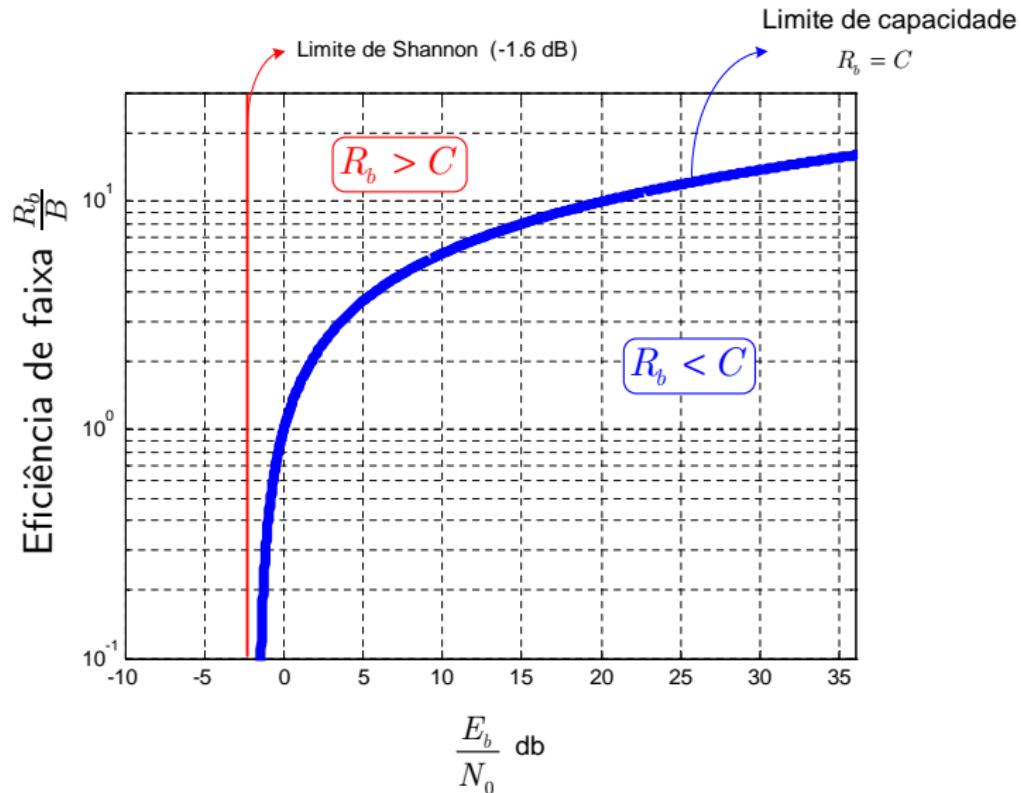
- De maneira equivalente pode-se definir a relação **energia do sinal por bit pela densidade espectral de potência** em termos da **eficiência de faixa** como

$$\frac{E_b}{N_0} = \frac{2^{C/B} - 1}{C/B} \quad (34)$$

- Eficiência de faixa  $\frac{C}{B}$

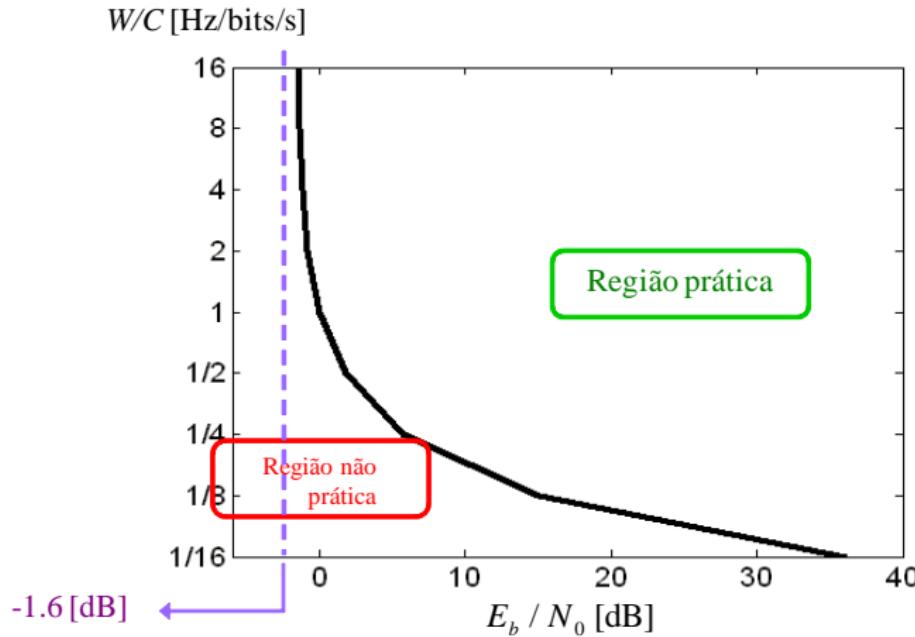
# Implicações do teorema da capacidade de informação

## Curva de eficiência de faixa - limite de Shannon



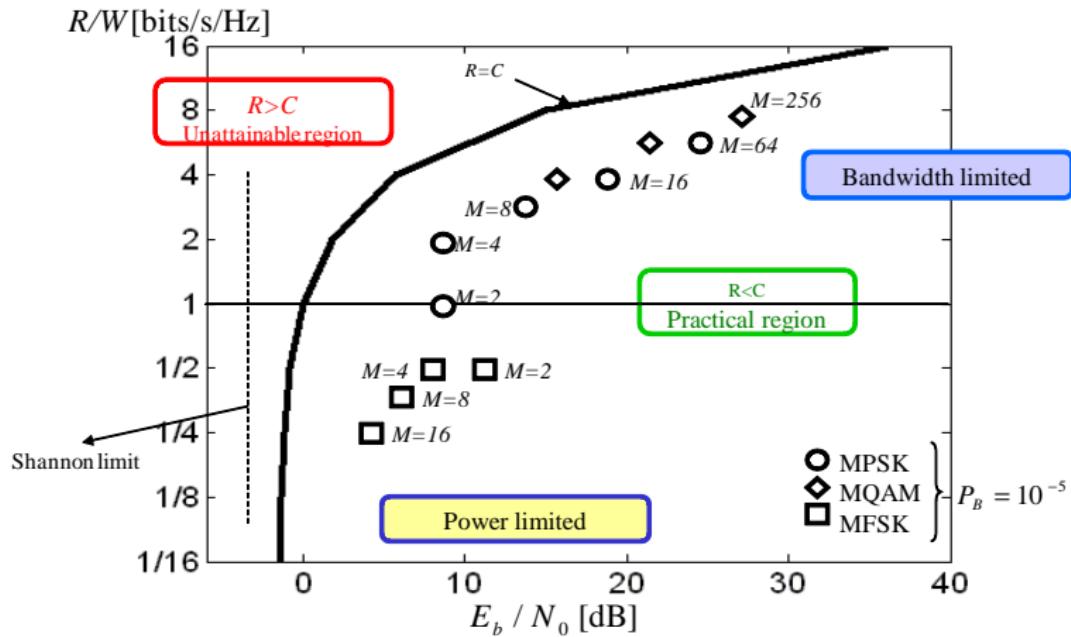
# Códigos Turbo (*Turbo Codes*)

Limite de Shannon



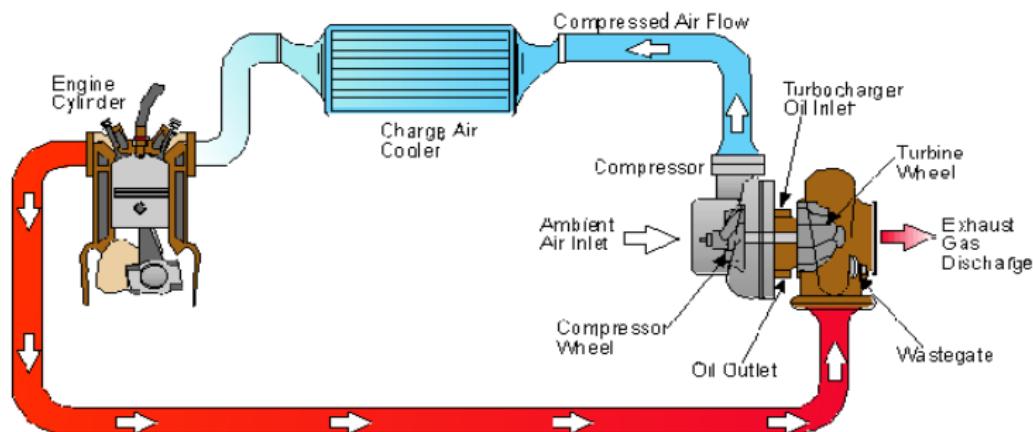
# Códigos Turbo (*Turbo Codes*)

## Limite de Shannon



# Códigos Turbo (*Turbo Codes*)

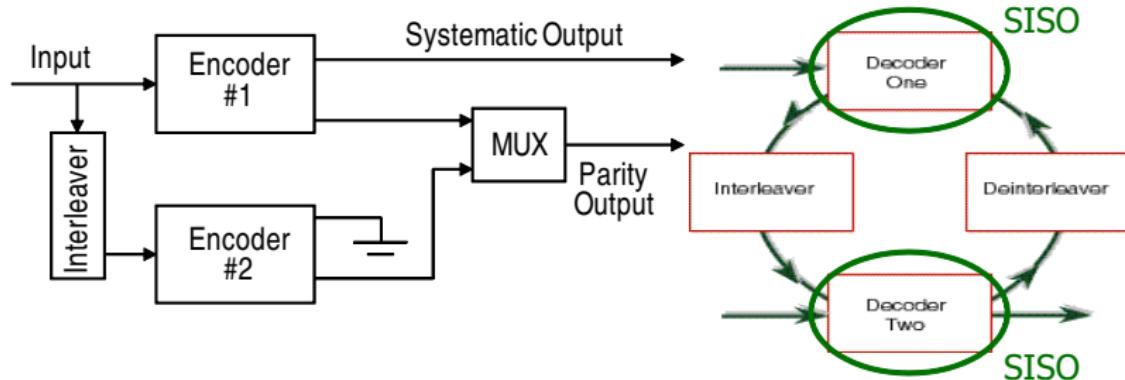
- ▶ Concatenação de dois RSCs em paralelo
  - ▶ recursividade: realimentação
  - ▶ sistemático: uma das saídas é a própria entrada
- ▶ Turbo codes: originário da engenharia mecânica: motores turbo - método de sobrealimentação



# Códigos Turbo (*Turbo Codes*)

- ▶ Codificador

- ▶ Decodificador



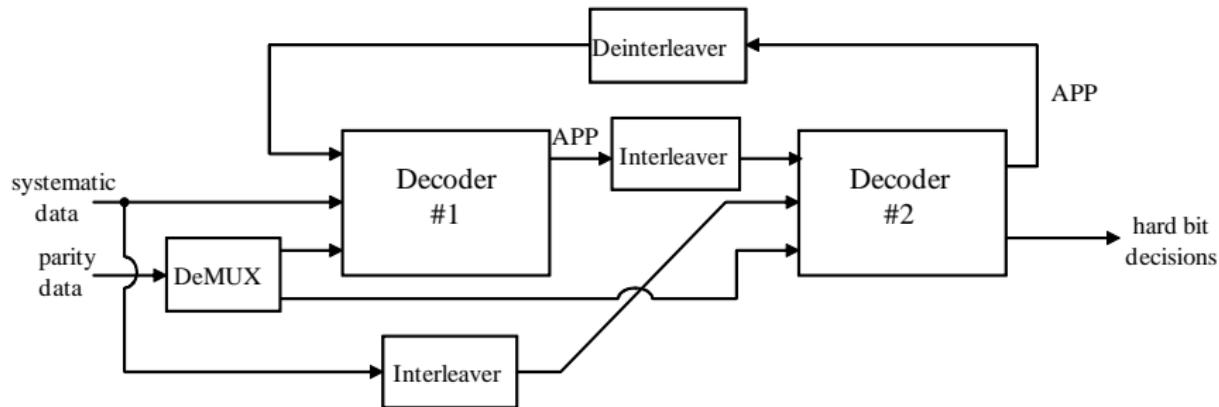
- ▶ Taxa do Código

$$\frac{1}{R} = \frac{1}{R_1} + \frac{1}{R_2} - 1 \quad (35)$$

- ▶ Processo iterativo

# Códigos Turbo (*Turbo Codes*)

## Decodificação Iterativa



- ▶ Existe um decodificador para cada codificador
- ▶ Cada decodificador estima a probabilidade a posteriori (APP) da cada bit
- ▶ As APPs são usadas com probabilidade a priori pelo outro decodificador no processo de decodificação iterativa
- ▶ O processo iterativo continua até um número estipulado de iterações
- ▶ O desempenho geralmente melhora de iteração-a-iteração

# Decodificação Iterativa

## Teorema de Bayes

$$P(A|B)P(B) = P(B|A)P(A) = P(A, B) \quad (36)$$

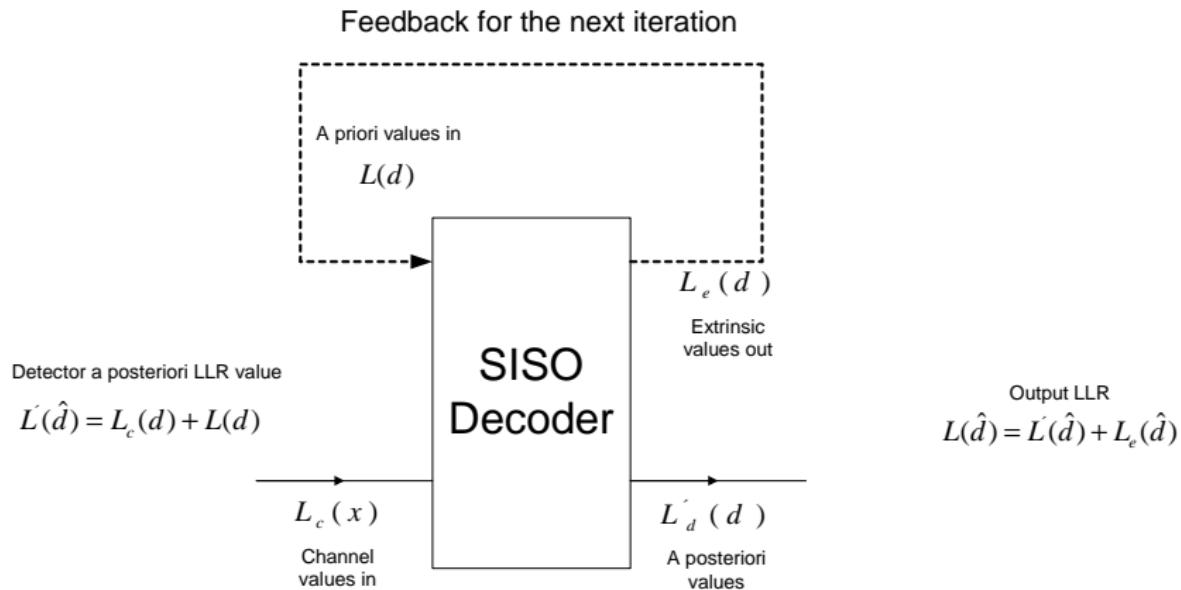
$$P(A|B) = \frac{P(B|A)P(A)}{P(B)} \quad (37)$$

- ▶  $P(A|B)$  - A posteriori probability (APP)
- ▶  $P(B|A)$  - conditional probability
- ▶  $P(A)$  - a priori probability

# Códigos Turbo (*Turbo Codes*)

Decodificação iterativa - ilustração

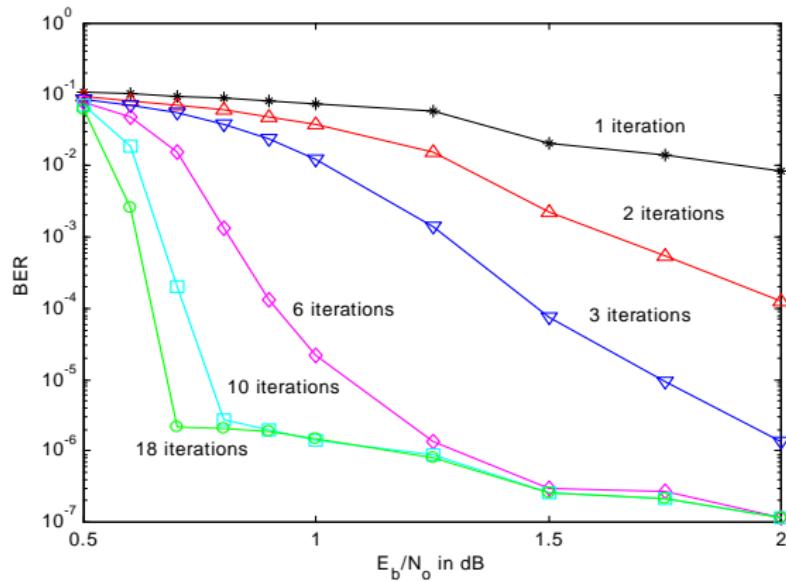
- ▶ Estrutura/princípio da decodificação



# Códigos Turbo (*Turbo Codes*)

## Decodificação iterativa - Desempenho

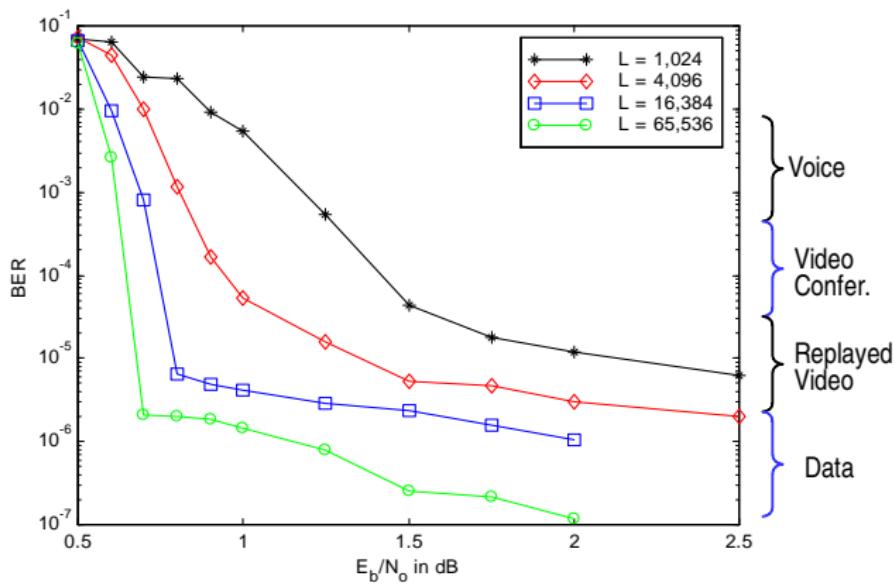
- $K = 5$
- $R = 1/2$
- $L = 2^{16} = 65,536$



# Códigos Turbo (*Turbo Codes*)

## Decodificação iterativa - Desempenho

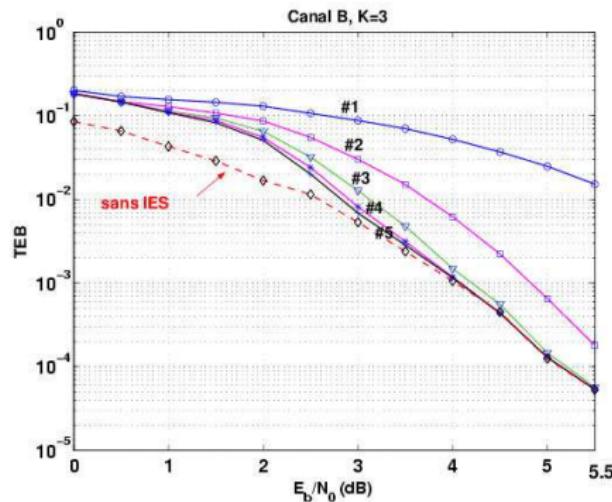
- $K = 5$
- $R = 1/2$
- 18 iter.



# Equalização Turbo (*Turbo Equalization*)

## Decodificação iterativa - Desempenho

- ▶ Canais : B [Proakis]
- ▶ Turbo Codes
- ▶ BPSK e 4096 depth interleaving



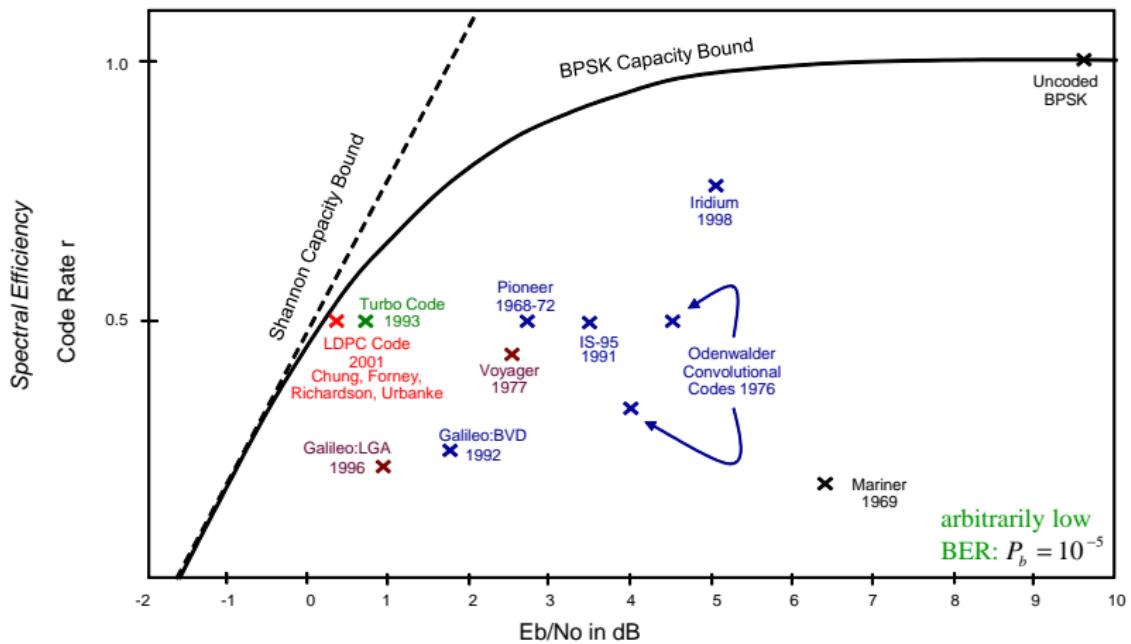
# Códigos Turbo

## Fatores de desempenho

- ▶ Complexidade vs. desempenho
  - ▶ Algoritmo de decodificação
  - ▶ Número de iterações.
  - ▶ constraint length codificador
- ▶ Latency vs. desempenho
  - ▶ Tamanho do frame
  - ▶ Projeto *Interleaving*
  - ▶ Padrão do punctionamento
  - ▶ Terminação da treliça item Taxa do código total

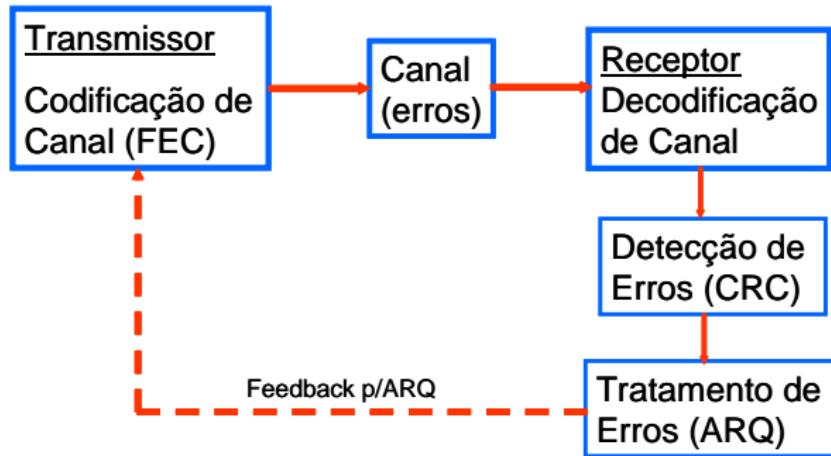
# Códigos Turbo

## Comparação



# Detecção e Tratamento de Erros

## Introdução



- ▶ Códigos de paridade
- ▶ Baixa capacidade de detecção

# CRC (*Cyclic Redundancy Check*)

## Procedimentos

- ▶ Representação binário-polinomial

$$1011 \rightarrow 1 * x^3 + 0 * x^2 + 1 * x + 1 = x^3 + x + 1$$

- ▶ Procedimento (Tx)

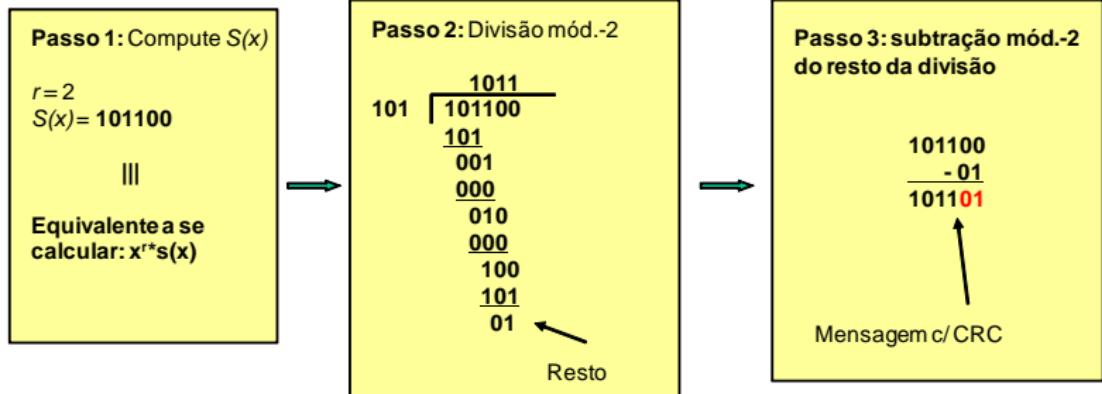
1. Seja  $r$  o grau do código polinomial  $g(x)$ . Adiciona-se  $r$  bits zero ao final da string de bits a ser transmitida. A nova seqüência é chamada  $S(x)$
2. Divida  $S(x)$  pelo polinômio gerador do código usando divisão módulo-2.
3. Subtraia o resto da divisão  $S(x)/g(x)$  usando subtração módulo-2.

- ▶ O resto são os bits de paridade da mensagem a ser transmitida

# CRC (Cyclic Redundancy Check)

## Codificação Exemplo

- Mensagem:  $1011 \rightarrow 1 * x^3 + 0 * x^2 + 1 * x + 1 = x^3 + x + 1$
- Código Polinomial:  $x^2 + 1$  (101)



# CRC (Cyclic Redundancy Check)

## Decodificação Exemplo

### ► Procedimento (Rx)

1. Seja  $n$  o tamanho da mensagem c/ os bits de CRC
2. Divida a mensagem pelo código polinomial  $g(x)$ . Se o resto de tal divisão for zero, não existe erro ou o erro é não detectável.

Mensagem c/CRC  
( $n=6$ ):

101101  
↓  
101

Mensagem original

Caso 1:

101		101101
		<u>101</u>
		001
		<u>000</u>
		010
		<u>000</u>
		101
		<u>101</u>
		00

Resto = 0  
(Erro não detectado)

Caso 2:

101		101001
		<u>101</u>
		000
		<u>000</u>
		000
		<u>000</u>
		001
		<u>000</u>
		01

Resto = 1  
(Erro detectado)

# CRC (*Cyclic Redundancy Check*)

## Códigos Usuais

<i>Código</i>	<i>Polinómio gerador</i>
CRC-12	$p^{12} + p^{11} + p^3 + p^2 + p + 1$
CRC-16	$p^{16} + p^{15} + p^2 + 1$
CRC-32 (Ethernet)	$p^{32} + p^{26} + p^{23} + p^{22} + p^{16} + p^{12} + p^{11}$ $+ p^{10} + p^8 + p^7 + p^5 + p^4 + p^2 + p + 1$
CRC-CCITT (norma X25)	$p^{16} + p^{12} + p^5 + 1$

*CRC-16*, por exemplo, significa que o número de bits de paridade é 16.

# CRC (*Cyclic Redundancy Check*)

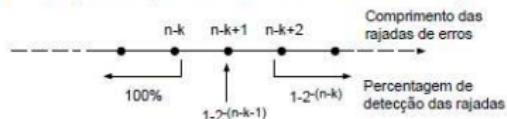
## Capacidade de Detecção de Erros

- ▶ Os códigos  $(n, k)$  binários são capazes de detectar os seguintes padrões de erro:
  1. Todas as rajadas de erros CRC de comprimento  $n - k$  ou menor
  2. Uma fração,  $1 - 2^{-(n-k-1)}$ , de rajadas de comprimento  $n - k + 1$
  3. Uma fração,  $1 - 2^{-(n-k)}$ , de rajadas de comprimento maior que  $n - k + 1$
  4. Todas as combinações de  $d_{min} - 1$ , ou menos, erros, como sempre
  5. Todos os padrões de erro com um número ímpar de erros se o polinômio gerador do código,  $g(p)$ , tiver um número par de coeficientes não nulos

# CRC (*Cyclic Redundancy Check*)

## Capacidade de Detecção de Erros

Percentagem de rajadas de erros detectadas com códigos CRC



Tipo de erros	CRC-12	CRC-16 CRC-CCITT	CRC-32
Rajadas de comprimento $\leq n - k$	100%	100%	100%
Rajadas de comprimento $n - k + 1$	99,951%	99,997%	99,99999995%
Rajadas de comprimento $> n - k + 1$	99,976%	99,998%	99,99999998%

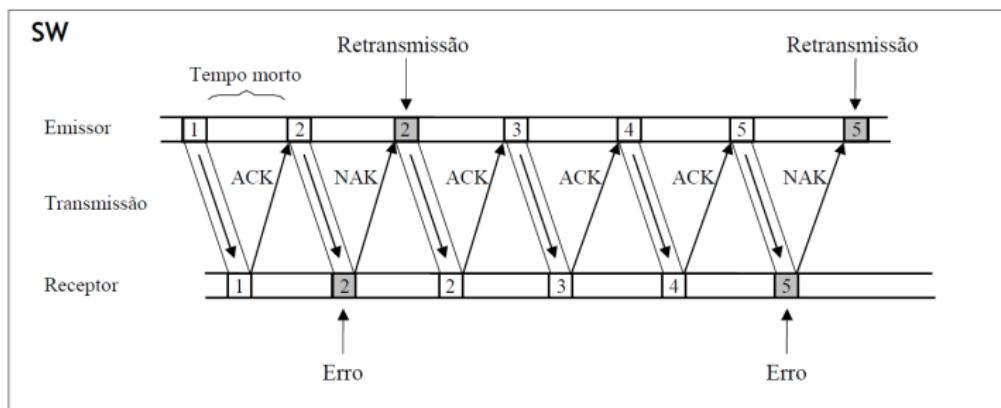
# Tratamento de Erros

- ▶ **Frame Erasure**: apaga quadros errados na transmissão de voz
- ▶ ARQ: identificado um bloco ou pacote com erro, o receptor pede a retransmissão do mesmo ao transmissor.
- ▶ **Stop and Wait ARQ** o mais simples, o transmissor espera uma mensagem de ACK após cada transmissão; se um erro for detectado o receptor envia um NAK e procede-se a uma retransmissão; um timer no transmissor também pode disparar o processo de retransmissão; novas mensagens devem ser armazenadas em um buffer no transmissor até o ACK ser recebido.
- ▶ **Go Back N ARQ** o transmissor continua transmitindo novas mensagens até que um NAK específico é recebido, identificando qual mensagem esteve em erro. O transmissor reinicia as transmissões de todas as mensagens a partir daquela errada.
- ▶ **ARQ Seletivo** um protocolo mais complexo que possui um buffer no receptor também. O NAK é específico de uma mensagem errada e o transmissor re-envia apenas aquela mensagem. O receptor rearranja a seqüência de pacotes após a recepção correta da mensagem retransmitida.

# Tratamento de Erros

## Stop-and-wait ARQ

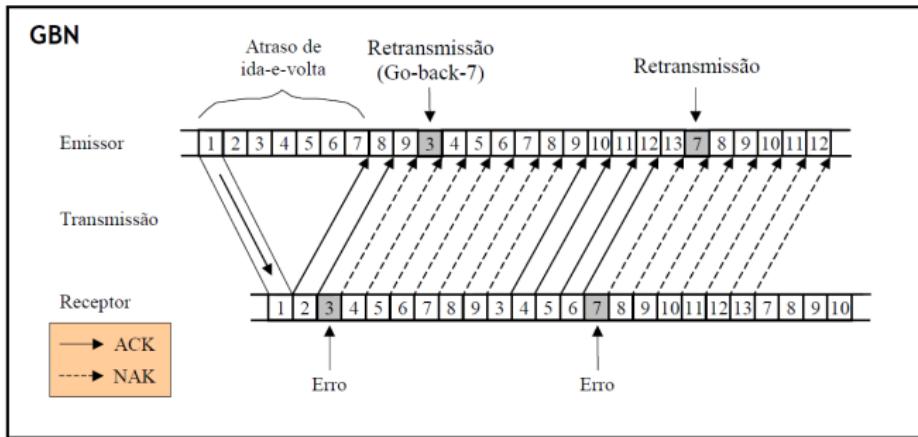
### Stop and Wait



# Tratamento de Erros

## Go-Back N

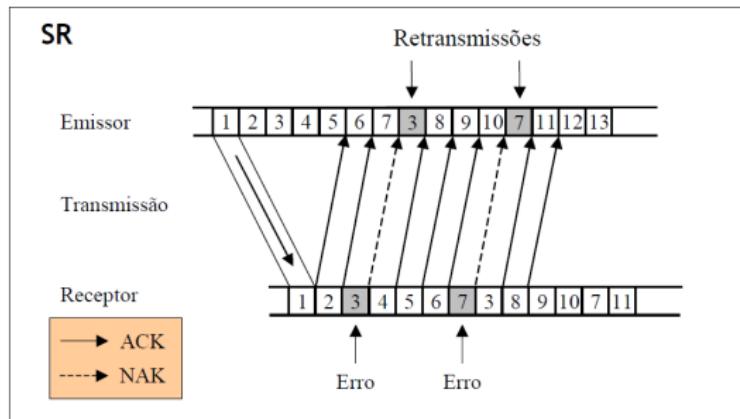
Go-back-N (com  $N = 7$ )



# Tratamento de Erros

## Selective-repeat ARQ

### Selective Repeat



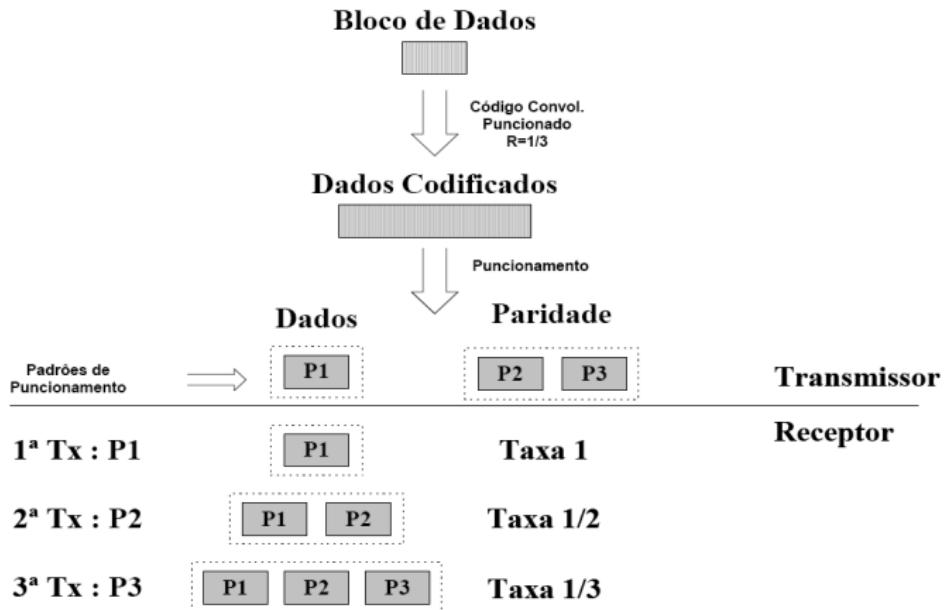
# Tratamento de Erros

## ARQ Híbrido

- ▶ Combinação de FEC e ARQ para melhorar desempenho  
Implementações possíveis:
  - ▶ **Tipo I** inclui apenas a FEC com capacidade de corrigir alguns blocos em erro.
  - ▶ **Tipo II - Redundância Incremental:** na detecção de um erro, o transmissor envia apenas um bloco de redundância adicional que será acrescido ao bloco recebido (maior redundância=maior proteção), e uma nova tentativa de detecção correta ocorrerá; o processo se repete até a detecção correta do bloco.
  - ▶ **Tipo III - Soft Combining:** Na detecção de um erro, o bloco inteiro é reenviado e combinado com o bloco recebido em erro, resultando num bloco com maior probabilidade de acerto (maior SINR devido a diversidade temporal); o processo continua sucessivamente até o acerto do bloco.
- ▶ Fortemente associada à adaptação de enlace. A taxa de codificação da primeira tentativa de transmissão definirá a necessidade de novas retransmissões.

# Tratamento de Erros

## ARQ Híbrido Tipo II



# Fim do Tópico

End of File!