Departamento de Engenharia de Eletrónica e Telecomunicações e de Computadores

Comunicação Digital – Módulo 2

Autores: 50536 Fábio Silva
50553 Bruno Raposo

Relatório para a Unidade Curricular de Comunicação Digital da Licenciatura em Engenharia Informática e de Computadores

Docente: Artur Ferreira

08 – 06 – 2024

<< Esta página foi intencionalmente deixada em branco >

Índice

Exercício 1

Neste exercício, é proposta a **simulação da transmissão de ficheiros sobre o BSC implementado** no último exercício do módulo anterior. Adicionalmente, os testes realizados ao BSC recorrerão a **3 configurações distintas**, sendo estas:

i. Ausência de códigos de controlo de erros.
ii. Código de repetição (3,1), em modo correção.
iii. Código de Hamming (7,4), em modo correção.

Para cada configuração serão apresentados **4 exemplos com valores de *p* crescentes**, onde, para cada exemplo, serão apresentados os **valores BER e BER'**, tal como o **número de palavras diferentes** entre o ficheiro recebido e o ficheiro original.

Códigos de controlo de erros

Os códigos de controlo de erros têm como função a deteção e, por vezes, a correção de erros que possam surgir em uma mensagem resultante de uma transmissão sujeita a erros. Esta deteção e correção são obtidas através da introdução de redundância na mensagem original, isto é, na adição de bits adicionais aos bits originais da mensagem.

Os códigos de controlo de erros lecionados em aula são: o código de repetição, o código de bit paridade par, o código de Hamming e o CRC (Cyclic Redundant Check). Destes códigos apresentados, recorreremos ao código de repetição e ao código de Hamming na resolução deste exercício, ambos em modo de correção.

Todos os códigos apresentados acima, com exceção do CRC, pertencem a um conjunto de códigos de controlo de erros designado de códigos lineares de bloco. Alguns conceitos importantes sobre este conjunto são:

- Cada *bloco*, isto é, cada conjunto de bits aos quais vão ser adicionados bits redundantes, de *k* bits de mensagem origina uma palavra de código com *n* bits (*k* bits da mensagem + *q* bits redundantes);
- O vetor nulo pertence ao código;
- A soma modular de quaisquer duas palavras do código resulta em uma outra palavra do código;
- Uma propriedade destes códigos é a distância mínima (dmin) que consiste no número de bits distintos entre palavras de código diferentes;
- O limite de bits errados que a mensagem pode ter até o código não ser capaz de indicar que ocorreu um erro na transmissão pode ser determinado através da seguinte fórmula:

$$l \leq \text{dmin} - 1$$

Figura 1 – Fórmula de determinação do limite máximo de bits errados para existir deteção de erros na transmissão (l = nº de bits errados; dmin = distância mínima)

- Por fim, o limite de bits errados até o código não ser capaz de corrigir qualquer erro quando detetado pode ser determinado através da seguinte fórmula:

$$t \leq \left\lfloor \frac{\text{dmin}-1}{2} \right\rfloor$$

Figura 2 – Fórmula de determinação do limite máximo de bits errados para existir correção de erros na transmissão (t = nº de bits errados; dmin = distância mínima)

Ausência de código de controlo de erros - Alínea (i)

Esta configuração do BSC tem como objetivo simular a transmissão de um ficheiro com a obra “Alice e o País das Maravilhas”, sujeita a erros, **sem recorrer a nenhum código de controlo de erros**.

Para a realização desta simulação recorreu-se a implementações realizadas no módulo anterior, nomeadamente, o BSC e a função de contagem de erros entre a mensagem recebida e a mensagem original.

A seguinte figura ilustra os resultados obtidos de 4 exemplos de simulação distintos com valores de *p* iguais a 0,001; 0,01; 0,1 e 0,5, respetivamente.

```
Example 1
Number of bits:  1187856
BER:  0.0
Number of wrong bits:  0
----------------------------
Example 2
Number of bits:  1187856
BER:  0.009187982381702833
Number of wrong bits:  10914
----------------------------
Example 3
Number of bits:  1187856
BER:  0.09933190555084118
Number of wrong bits:  117992
----------------------------
Example 4
Number of bits:  1187856
BER:  0.49941912150967793
Number of wrong bits:  593238
```

Figura 3 - Resultados da simulação sem código de controlo de erros

Uma vez que estes exemplos de simulação não recorrem a nenhum código de controlo de erros, não existe valor de BER' a ser apresentado.

Ao observarmos a imagem, é possível compreender que quanto maior for o valor de *p*, ou seja, o BER, maior será o número de erros observados no ficheiro.

As seguintes figuras mostram um excerto dos ficheiros resultantes dos exemplos de transmissão do ficheiro com a obra "Alice e o País das Maravilhas", pelo nosso BSC.

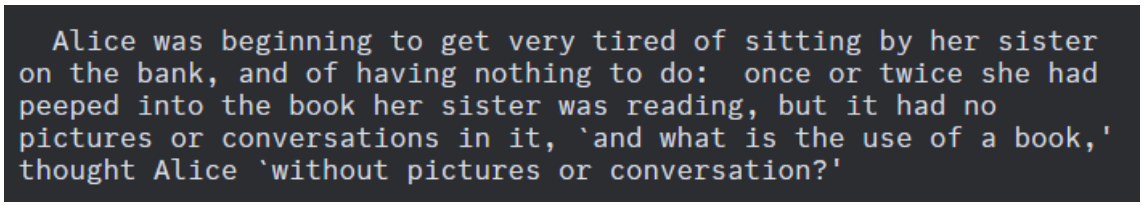

Figura 4 - Excerto do ficheiro resultante de uma transmissão ausente de controlo de erros com BER = 0,001

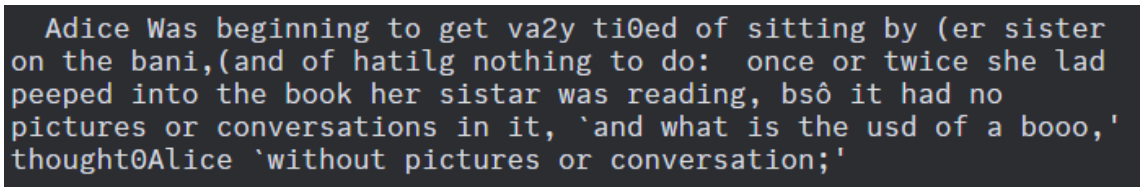

Figura 5 - Excerto do ficheiro resultante de uma transmissão ausente de controlo de erros com BER = 0,01

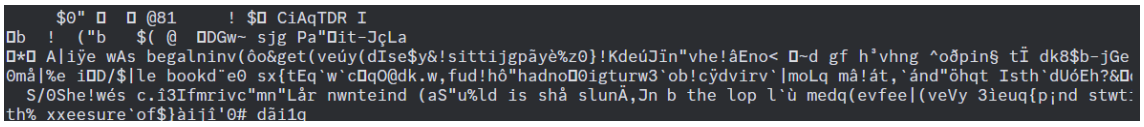

Figura 6 – Excerto do ficheiro resultante de uma transmissão ausente de controlo de erros com BER = 0,1

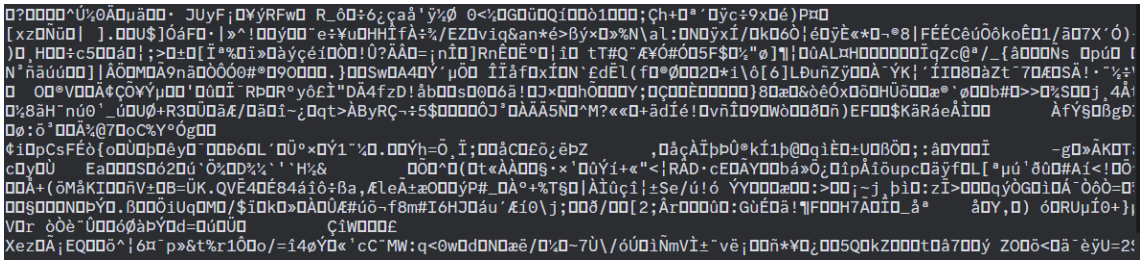

Figura 7 - Excerto do ficheiro resultante de uma transmissão ausente de controlo de erros com BER = 0,5

Código de repetição - Alínea (ii)

O objetivo desta configuração do BSC consiste em, novamente, simular a transmissão da obra “Alice e o País das Maravilhas”, **recorrendo ao código de repetição (3,1) em modo de correção**, como forma de deteção e correção de erros durante a transmissão.

O código de repetição (3,1) consiste em um **código de controlo de erros** que, de forma a conseguir detetar uma possível troca de bits na transmissão, **adiciona a cada bit da mensagem duas repetições do mesmo**, antes desta ser transmitida. Com esta codificação, uma mensagem do tipo “0 1 1 0” resultará no código “000 111 111 000”.

Após a transmissão da mensagem codificada pelo BSC, o descodificador irá verificar, para cada **conjunto de 3 bits**, se algum destes foi modificado, indicando que houve um erro de transmissão quando **todos os bits do conjunto não são iguais**. Desta forma, este código de controlo de erros é capaz de **detetar erros de 1 e 2 bits na mensagem codificada**. Contudo, a sua **capacidade de correção de erros já se encontra limitada a apenas 1 bit**. Isto porque, após a transmissão da mensagem codificada, o descodificador vai verificar os 3 bits de cada conjunto e descodificar, os mesmos, no bit que se encontra em maioria, descodificando de forma errada, conjuntos de bits que tenham sofrido alterações em 2 ou 3 dos seus bits. Com esta descodificação, um código sujeito a erros do tipo “001 101 001 000” resultará na mensagem “0 1 0 0”.

Para a realização desta simulação recorreu-se, novamente, a implementações realizadas no módulo anterior, nomeadamente, o BSC e a função de contagem de erros entre a mensagem recebida e a mensagem original.

A seguinte figura apresenta os resultados obtidos de 4 exemplos de simulação distintos com valores de *p* iguais a 0,001; 0,01; 0,1 e 0,5, respetivamente.

```
Example 1
- Without Encoding
Number of bits:  1187856
BER:  0.0
Number of wrong bits:  0
- With Encoding
Number of bits:  1187856
BER':  0.0
Number of wrong bits:  0
---------------------------
Example 2
- Without Encoding
Number of bits:  1187856
BER:  0.008895859430772753
Number of wrong bits:  10567
- With Encoding
Number of bits:  1187856
BER':  0.00023740251343597204
Number of wrong bits:  282
```

```
Example 3
- Without Encoding
Number of bits:  1187856
BER:  0.0995028016863997
Number of wrong bits:  118195
- With Encoding
Number of bits:  1187856
BER':  0.027813977451812 34
Number of wrong bits:  33039
---------------------------
Example 4
- Without Encoding
Number of bits:  1187856
BER:  0.5002803370105467
Number of wrong bits:  594261
- With Encoding
Number of bits:  1187856
BER':  0.49960011987985076
Number of wrong bits:  593453
```

Figura 8 – Resultados da simulação com código de repetição (3,1)

Com as imagens apresentadas acima conseguimos perceber que:

- A implementação com o código de repetição irá resultar em um **menor número de erros na mensagem** recebida;
- Quanto maior o valor de *p*:
  - **Maior será o número de erros na mensagem recebida**, quer a transmissão aplique o código de repetição, quer não;
  - **Menor será a diferença entre o número de bits errados da transmissão com código de repetição e o número de bits errados da transmissão sem código de controlo de erros**. Isto porque, quanto maior for o BER da transmissão, maior será a probabilidade de cada bloco de bits apresentar erros em 2 ou 3 bits, impossibilitando a correção dos mesmos.

As figuras seguintes demonstram um excerto dos ficheiros resultantes dos exemplos de transmissão com código de repetição (3,1), da obra “Alice e o País das Maravilhas”, pelo nosso BSC.

```
  Alice was beginning to get very tired of sitting by her sister
on the bank, and of having nothing to do:  once or twice she had
peeped into the book her sister was reading, but it had no
pictures or conversations in it, `and what is the use of a book,'
thought Alice `without pictures or conversation?'
```

Figura 9 - Excerto do ficheiro resultante de uma transmissão com código de repetição (3,1) e BER = 0,001

```
  Alice was beginning to get very tired of sitting by her sister
on the bank, and of having nothing to do:  once or twice she had
peeped into the book her sister was reading, but it had no
pictures or conversations in it, `and what is the use of a book,'
thought Alice `without pictures or conversation?'
```

Figura 10 - Excerto do ficheiro resultante de uma transmissão com código de repetição (3,1) e BER = 0,01

```
□" Álice was□"eginfmngto get veri tired îf {atting by heò"shsueb
On ôhe(baNk, and og0heving nithing To dO:  once Or twice sHe had
reepgdint/ the booj$hev sister Was reaäing, but hv jid no
0icöures¤or$conversatiols in it, `!jd what!is tèe use(of a Book,'
thougHt Alice `withou| picturdó or convessatmgn?'□□  Sn she wes consit%ring in hår own mind
for!thEhot day made har feel very ó,eePy and stupid), whe4her
thg pleasure of making a daisy-shaif would"be wozth the vsoUbld
of!getting }p and pic{ing the daióies¬!vhun suddenlya!Shéte
```

Figura 11 - Excerto do ficheiro resultante de uma transmissão com código de repetição (3,1) e BER = 0,1

```
Õ¾Yµ□□°ðf#µ9□eãjw□Åâü Æ[ «7□_åã_~C□yhÍáÊú¥□Ú□¯5Ú}        □é.¯¿òóÈ®;□J□□è□ÿ±áÇ°l¯□□È□F*ð □□·üOe□q?P□þøyÁ¼¡Ìo°vÿ□□□
3c_q□É□§,       Zk□□ó□□18-Þ□¥ª}□®□k□ÔKÎ×A□□¾□ñ÷□J□□V□□JRwiy□ÿi□£xgF8Ìai2<□åþ□□~9"háúYK□í3Û×cÊ□Qj«ªÉ□Å¶ÝòÙÎ□$□WAò
>tÔ□Þe,¾j□^sTi□:i□ù»°¸ë=MýÛu□wutnÙ□X□Ô±%□{t~ÊBÊ&ÝJ□Ø Ý □âË□Æ□p□□q2JÄGí%|□□□ö_Ó[! □ûA□
□□¨□d□ÁN$ê¢;ýÞ°þX»ÒwÙ[ÝÅ□MJRQl¨øqé=□3□□Lòé□ó □ßþ□à!□¬sg»□<æJ(á□vÅ□&ÓáM%¨äÂª□□rí\-F□□□?Ãú-□Ôg□%¾®óêoÑw¢ð¨u+^□□«äW
 ÓY®&FÅ□µ®-□ Ë□ðN¯oDv□®NuFQi¨¶<4ñÕ<±!mMèYN`@□h□PÒ0□HL□Ô□"Å+X;□□□°²U½MTr□□°□Î÷öÀU□Sçô
e□a□Hx=□ª□□□Mà□□¢>v□ïZó□æ□ß5TCÜó□Á□¼□Çi□¯ØUáÒxqg        tè□2qázwôï²□è□:□□É□□1G=í|□<i«□¾□ú8□¸□`Ã□□□A¦½E□3
N□□□~QQýÖÎ2¤pl□=ü□°Ëîoêä1/}%Ð□TN□Îòk□U4□4□¶ZÏ□æÏ\Ý□□U«%c□ïî□(Ø¸Ó µ□□-¼Ô4□\□□□À□□D'□nÔ□ç8Ëw□J÷L,□åzU:í[°□Wçæ¸R□·½È
□□°□1□îo{□±qéØ}□"R□pî□□fåö□ .□|□2□□×àÂ□4ç□ÁQ<n#□|íÁ5ï□T+´°□C«□□Ü¯u¯}³M□Å      â□ö|□D□U□på
¿µ¯□¢À¤GÜMl~ê□ñ ã□ê□8¯]£è,\r¾□Á□¬□é□ÇÛ□*¯□E□gM(),ªbp□□SÔuIí□¶□^NvÙ·□âI□|□□ë□uÕqQ□□;°[□□□□□ê<7GÎI8·2¾Iñ4'□g□□Þ□¯
ónÐ□ÖPÍZDÖ[□5_□2□*□&ì□Kl□íZF□'~□¤Ag□~□ìn□VÀ*µç□□H¨□□cÎÚG¦□ð%úäö¶ï´Îù□;□{¿ÙÒ□□`□□h%i;'□ïJº□õ `&□□çòxßãaöâ□à□□ÆA□æ{
□R5Åö□NáêÐz¬□□Íª I%□
```

Figura 12 - Excerto do ficheiro resultante de uma transmissão com código de repetição (3,1) e BER = 0,5

Código de Hamming - Alínea (iii)

Nesta configuração do BSC pretende-se simular, de novo, a transmissão da obra “Alice e o País das Maravilhas”, **recorrendo ao código de Hamming (7,4) em modo de correção**, como forma de deteção e correção de erros durante a transmissão.

O código de Hamming (7,4) consiste em um outro **código de controlo de erros** que, de forma a conseguir detetar erros na transmissão, **realiza 3 equações de forma a gerar os 3 bits de paridade do código**, antes desta ser transmitida. Estas equações são:

$$b_0 = m_1 \oplus m_2 \oplus m_3$$
$$b_1 = m_0 \oplus m_1 \oplus m_3$$
$$b_2 = m_0 \oplus m_2 \oplus m_3$$

Figura 13 - Equações de paridade

Uma vez que o cálculo dos bits de paridade é fixo, este código de controlo de erros apresenta apenas 16 palavras de código possíveis:

- 0000 000
- 0001 111
- 0010 101
- 0011 010
- 0100 110
- 0101 001
- 0110 011
- 0111 100
- 1000 011
- 1001 100
- 1010 110
- 1011 001
- 1100 101
- 1101 010
- 1110 000
- 1111 111

O código de Hamming é **capaz de detetar erros até 2 bits e corrigir erros de 1 bit**, isto porque **todos os seus códigos apresentam uma distância mínima igual a 3**.

Após a transmissão da mensagem codificada pelo BSC, o descodificador vai, para **cada conjunto de 7 bits**, aplicar, novamente, as equações de paridade para os bits da mensagem, comparando estes com os bits de paridade da mensagem resultante da transmissão. Caso esta comparação conclua que os bits são diferentes, então ocorreu um erro na transmissão da mensagem.

De forma a corrigir quaisquer erros de 1 bit na mensagem recebida, o descodificador vai recorrer a uma tabela de síndromas com o objetivo de detetar o bit errado e corrigir o mesmo. Esta tabela de síndromas, específica para este código de controlo de erros, consiste em uma tabela que relaciona uma codificação de 3 bits (a síndroma) com o número do bit da mensagem que se encontra errado, sendo que se a codificação for igual ao vetor nulo, então a mensagem não apresenta nenhum erro. A síndroma poderá ser determinada através da seguinte operação:

$$(e_0 \text{ XOR } r_0) + (e_1 \text{ XOR } r_1) + (e_2 \text{ XOR } r_2),$$

sendo ***e*** os bits resultantes da aplicação das equações de paridade nos bits de mensagem do código recebido e ***r*** os bits de paridade do código recebido.

A tabela de síndromas mencionada é ilustrada na seguinte figura:

Síndroma	Padrão de Erro	Observações
000	0000000	Ausência de erro
011	1000000	1.º bit em erro
110	0100000	2.º bit em erro
101	0010000	3.º bit em erro
111	0001000	4.º bit em erro
100	0000100	5.º bit em erro
010	0000010	6.º bit em erro
001	0000001	7.º bit em erro

Figura 14 – Tabela de síndromas específica para o código de Hamming (7,4)

Para a realização desta simulação recorreu-se, novamente, a implementações realizadas no módulo anterior, nomeadamente, o BSC e a função de contagem de erros entre a mensagem recebida e a mensagem original.

A seguinte figura apresenta os resultados obtidos de 4 exemplos de simulação distintos com valores de *p* iguais a 0,001; 0,01; 0,1 e 0,5, respetivamente.

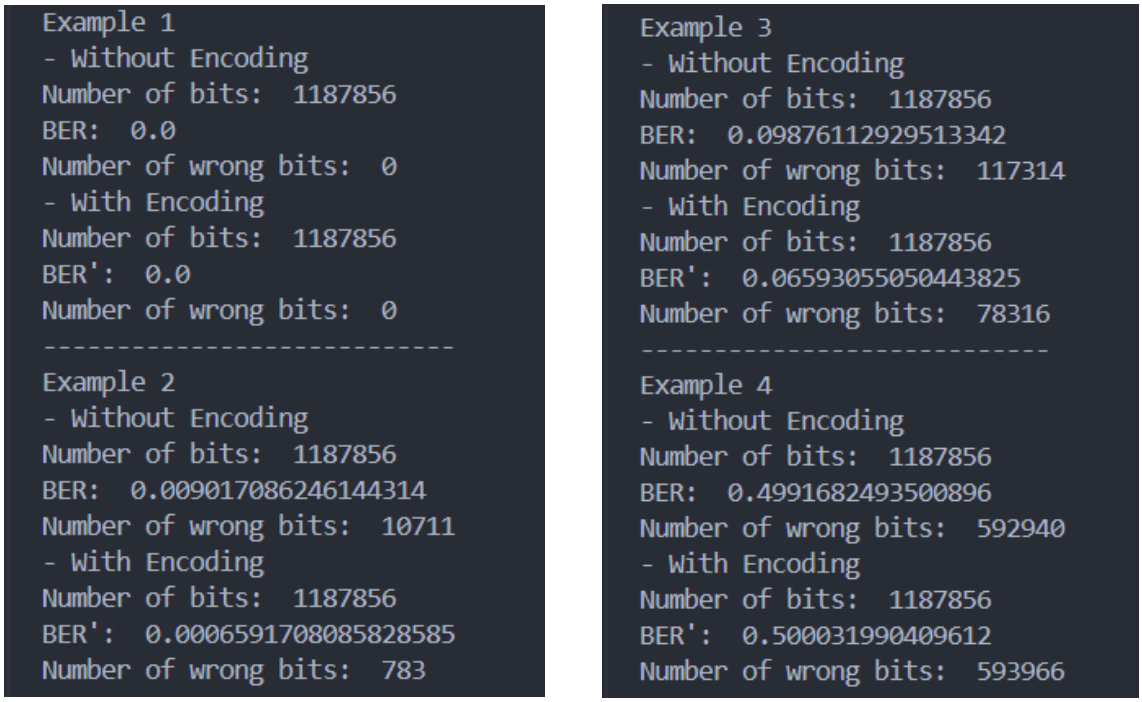

Figura 15 - Resultados da simulação com código de Hamming (7,4)

Com as imagens apresentadas acima conseguimos perceber que:

- A implementação com o código de Hamming irá resultar em um **menor número de erros na mensagem** recebida;
- Quanto maior o valor de *p*:
  - **Maior será o número de erros na mensagem recebida**, quer a transmissão aplique o código de Hamming, quer não;
  - **Menor será a diferença entre o número de bits errados da transmissão com código de Hamming e o número de bits errados da transmissão sem código de controlo de erros**. Isto porque, quanto maior for o BER da transmissão, maior será a probabilidade de cada bloco de bits apresentar erros em mais do que 1 bit, impossibilitando a correção dos mesmos.

As figuras seguintes demonstram um excerto dos ficheiros resultantes dos exemplos de transmissão com código de Hamming (7,4), da obra “Alice e o País das Maravilhas”, pelo nosso BSC.

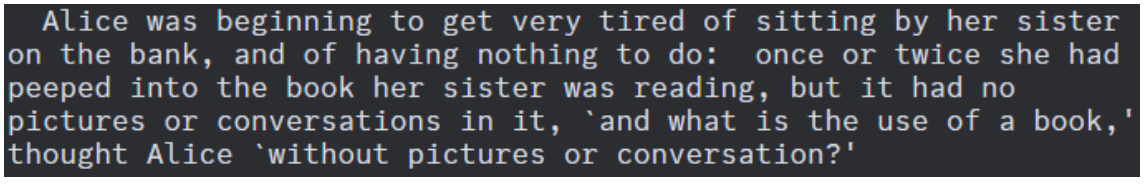

Figura 16 - Excerto do ficheiro resultante de uma transmissão com código de Hamming (7,4) e BER = 0,001

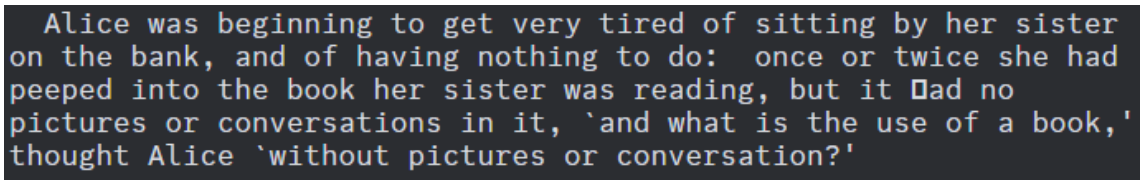

Figura 17 - Excerto do ficheiro resultante de uma transmissão com código de Hamming (7,4) e BER = 0,01

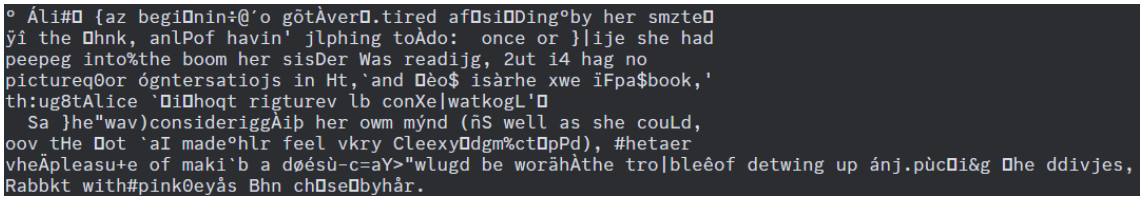

Figura 18 - Excerto do ficheiro resultante de uma transmissão com código de Hamming (7,4) e BER = 0,1

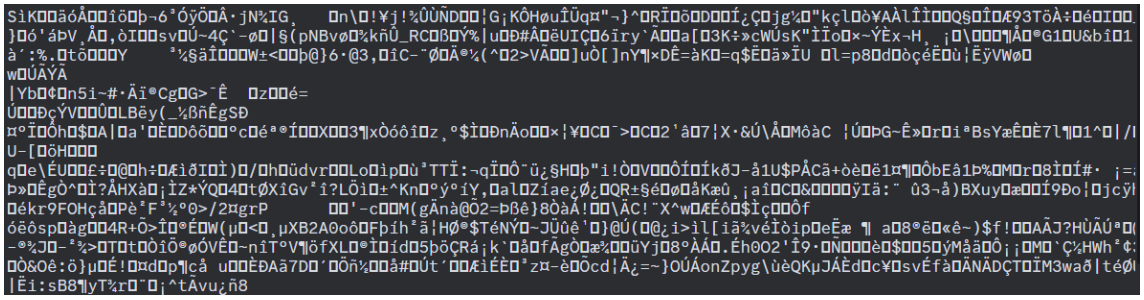

Figura 19 - Excerto do ficheiro resultante de uma transmissão com código de Hamming (7,4) e BER = 0,5

Exercício 2

Neste exercício, exploraremos os conceitos de **erros em rajada** e **deteção de erros com Cyclic Redundancy Check (CRC)**. Primeiro, abordaremos como os erros em rajada afetam a integridade dos dados transmitidos. Em seguida, discutiremos o uso do CRC para detetar erros em blocos de dados.

Erros em rajada (Burst) – Alínea (a)

Erros em rajada ocorrem quando vários bits consecutivos em uma unidade de dados são corrompidos durante a transmissão. Os **erros em rajada** são frequentes em sistemas de comunicação e armazenamento. Diferentemente dos erros de bit individuais, que afetam apenas um bit, os erros em rajada podem impactar sequências maiores de bits. Esses erros podem ser causados por interferências no sinal, ruídos impulsivos ou outras perturbações. Exemplos de situações reais:

1. **QR Codes:**

- Os **QR codes** são amplamente utilizados para armazenar informações, como URLs ou dados de contato.
- Durante a leitura de QR codes danificados ou mal impressos, erros em rajada são uma possibilidade.

2. **Transmissão de Vídeo Digital:**
   - Em sistemas de transmissão de vídeo digital, como a TV digital, os erros em rajada podem ocorrer devido a interferências no sinal, provocando por vezes distorção parcial do vídeo que está a ser transmitido.

Dentro deste contexto, existem vários códigos projetados para deteção e correção. No exercício 2, focar-nos-emos num deles: o **Cyclic Redundancy Check (CRC)**. Discutiremos mais à frente como o CRC é utilizado e de que forma é capaz de detetar os erros em rajada.

Na alínea (**a**), desenvolvemos uma função em **Python** que recebe uma sequência de bits, um valor (**L**) que representa o tamanho dos erros em rajada e um valor (**p**) que corresponde à probabilidade de inserção de erros em rajada na sequência inicial.

No diagrama de blocos representado pela **Figura 20**, apresentamos os blocos pelos quais o arquivo de texto passará. A única diferença em relação ao exercício 1 é que substituiremos o **canal BSC** por um **canal Burst**, onde os erros em rajada serão introduzidos.

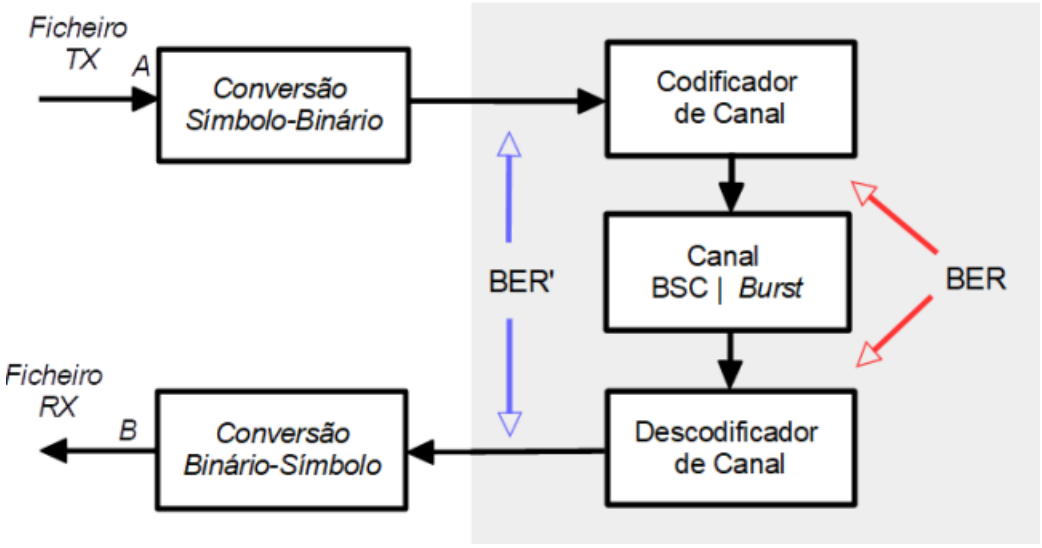

Figura 20 - Diagrama de blocos do canal

Para ilustrar a passagem por este canal de Burst, geramos quatro arquivos diferentes com base no mesmo texto. Em seguida, analisaremos como diferentes valores de (**L**) e (**p**) influenciam o conteúdo desses arquivos.

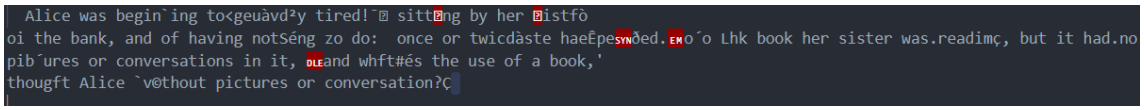

Figura 21 - Ficheiro gerado pela passagem pelo Burst com valores L=3 e p=0.01

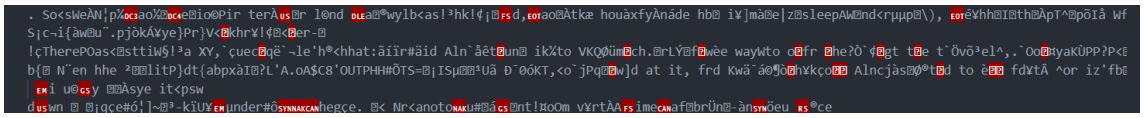

Figura 22 - Ficheiro gerado pela passagem pelo Burst com valores L=3 e p=0.1

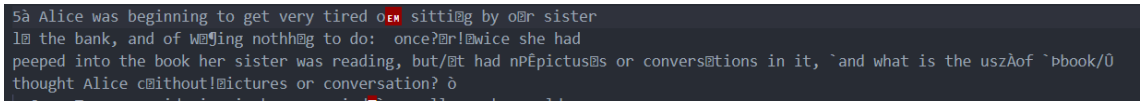

Figura 23 - Ficheiro gerado pela passagem pelo Burst com valores L=8 e p=0.01

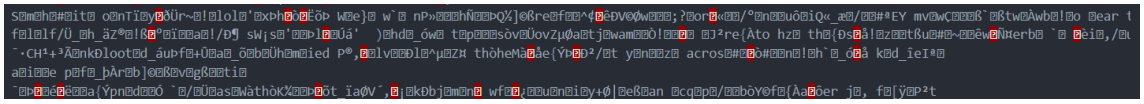

Figura 24 - Ficheiro gerado pela passagem pelo Burst com valores L=8 e p=0.1

É notável o impacto que os **erros em rajada** têm nos arquivos gerados. Mesmo com valores mais baixos de (**L**) e (**p**), o conteúdo inicial do texto foi significativamente afetado, como evidenciado na **Figura 21**. Além disso, na **Figura 24**, a leitura do texto ficou completamente ilegível.

É importante ressaltar que, no mundo real, a probabilidade de ocorrência de erros em rajada não é tão frequente quanto a que utilizámos para fins de teste. No entanto, estes exemplos permitem-nos compreender como erros desta dimensão influenciam as transmissões de dados.

Deteção de Erros em Rajada com CRC – Alínea (b)

Na alínea anterior, implementámos o modelo de canal que introduz **erros em rajada** com dimensão (**L**). Agora, vamos aprofundar como o **Cyclic Redundancy Check (CRC)** pode ser aplicado a esse modelo para detetar esses erros.

O **Cyclic Redundancy Check (CRC)** é uma técnica amplamente utilizada para deteção de erros em transmissões de dados. Os principais conceitos relacionados ao CRC são: [1]

1. **Síndroma (s(X))**: [1]
   - O descodificador, em modo de deteção, calcula a síndroma (s(X)).
   - Dado que (c(X) = m(X)g(X)), qualquer palavra de código é um fator do polinómio gerador.
2. **Palavra Recebida (y(X))**: [1]

- Seja (y(X) = c(X) + e(X)), onde (e(X)) é o padrão de erro.
- Se (e(X)) for nulo, a síndroma também será nula.
- Caso contrário, a síndroma será não nula e dependerá do valor de (e(X)).

3. **Divisão de Polinómios**: [1]
   - Na descodificação, realizamos a divisão de polinómios.
   - A síndroma é calculada para verificar se há erros.
   - Se a síndroma for nula, não há erros detetados.
   - Caso contrário, erros são identificados e dependem do valor de (e(X)).
4. **Capacidade de Deteção de Erros**: [1]
   - O CRC tem alta capacidade de deteção de erros, especialmente de **burst de erros** (rajadas de erros).
   - Identifica erros em sequências contíguas de bits, mesmo quando esses erros ocorrem em blocos maiores.
5. **Limitações**: [1]
   - O CRC não corrige erros; ele apenas os deteta.
   - Se a síndroma for não nula, o recetor sabe que ocorreram erros, mas não sabe exatamente quais bits estão errados.

No âmbito da alínea (**b**), desenvolvemos duas funções para deteção de erros em rajada usando o CRC. Uma função para **codificação** e outra para **descodificação**. O objetivo era calcular os bits de paridade da mensagem transmitida usando o polinômio gerador **CRC16**. Na descodificação, separamos a mensagem dos bits de paridade, recalculamos os mesmos e comparamos estes com os bits de paridade recebidos para verificar se eram iguais e se não houve troca de bits durante a transmissão. Utilizamos a biblioteca chamada **CRCCHECK** do **Python** para realizar o cálculo do CRC.

Para testar a deteção de erros usando o CRC, geramos três sequências de 1024 bits. Passámos cada sequência pelo canal de Burst conforme o diagrama de blocos da Figura 20. Em seguida, comparámos os bits de paridade recebidos com os bits de paridade calculados durante a descodificação. Vale a pena realçar que utilizámos uma taxa de erro de bit (BER) de 0,002 e um valor de L igual a 16 o que nos permitiu atingir os resultados representados nas figuras abaixo.

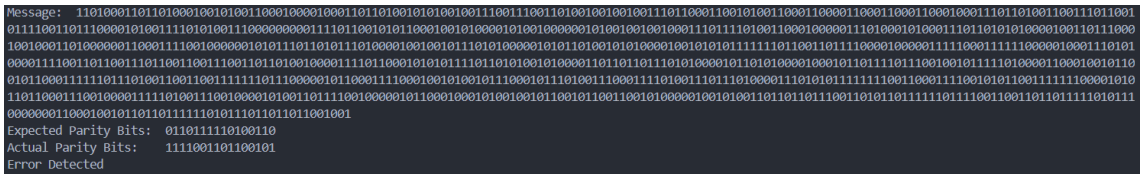

Figura 25 - Exemplo 1 da passagem pelo canal de burst e deteção de erros feito pelo CRC

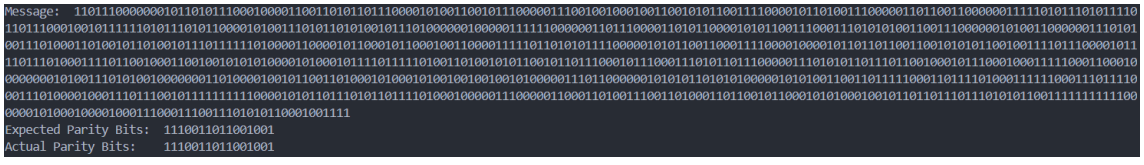

Figura 26 - Exemplo 2 da passagem pelo canal de burst e deteção de erros feito pelo CRC

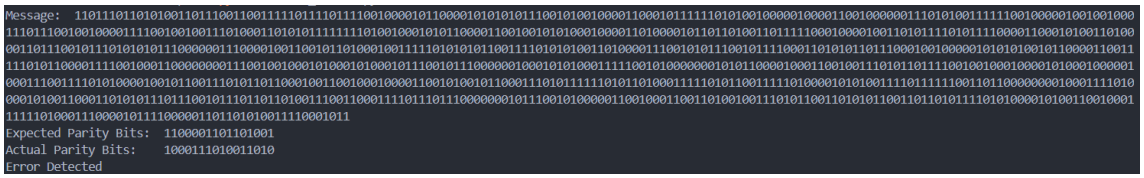

Figura 27 - Exemplo 3 da passagem pelo canal de burst e deteção de erros feito pelo CRC

Identificação de Erros Indetetáveis pelo CRC – Alínea (c)

Conforme mencionado anteriormente, uma das características notáveis do **Cyclic Redundancy Check (CRC)** é a sua habilidade de deteção de **rajadas de erros**. No entanto, existem 3 situações onde não é possível detetar erros, se o padrão de erro for múltiplo do polinómio gerador: [2]

1. **Situação 1**:
    - Para rajadas de erros com comprimento menor ou igual ao grau ((n - k)) do polinómio gerador (g(x)), não podem corresponder aos múltiplos do polinómio gerador e todos os padrões de erros são detetados. [2]
2. **Situação 2**:
    - Para rajadas de erros com comprimento ((n - k + 1)), existe apenas um padrão de erro múltiplo de (g(x)) que coincide exatamente com (g(x)).
    - A relação de rajadas de erros não detetadas é inversamente proporcional ao número de padrões de erro possíveis. [2]
3. **Situação 3**:
    - Para rajadas de erros com comprimento ((n - k + 2)), o número de padrões de erro diferentes aumenta, mas também aumenta o número de múltiplos de (g(x)).
    - A relação de rajadas de erros não detetadas permanece constante. [2]

Para demonstrar a ocorrência de um erro em rajada que não é detetado, iniciámos o processo por consultar a documentação da biblioteca **CRCCHECK**. O objetivo era identificar o polinómio gerador utilizado pelo CRC16 desta biblioteca. Identificámos que o polinómio gerador era $g(x) = x^{13} + x^5 + 1$.

Com o polinómio gerador(**4129**) identificado, procedemos à seleção de uma sequência de bits que fosse múltipla do polinómio e escolhemos “0001000000100010”(**8158**). Para provar que o erro não seria detetado, introduzimos bits errados na mensagem recebida para que também fosse um múltiplo do polinómio gerador. Assim, gerámos “0001000000100001000000”(**33032**).

Confirmámos o que era o objetivo e o resultado que obtivemos está visível na figura 28. Podemos ver que foram gerados os mesmos bits de paridade para ambas as mensagens, o que corresponde exatamente ao que era esperado. Por este motivo, o CRC não irá detetar que houve diferença nas mensagens enviadas e recebidas. Este exemplo demonstra uma das situações em que um erro em rajada não é detetado pelo CRC.

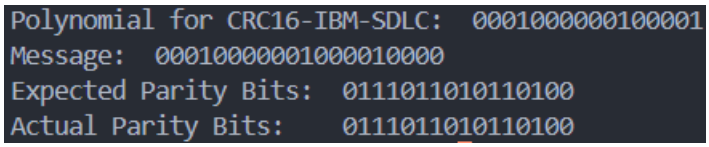

Figura 28 - Exemplo da não deteção de erros pelo CRC

Exercício 3

No Exercício 3, iremos aprofundar a nossa compreensão da plataforma *Arduino* e expandir o nosso conhecimento sobre os **Sistemas de Comunicação Digital (SCD**). Isto será alcançado através da simulação de uma transmissão de informação autêntica, estabelecida por uma **conexão USB** entre o *Arduino* e a nossa máquina.

Adicionalmente, iremos incorporar um **mecanismo de deteção de erros** na nossa comunicação. Para isso, utilizaremos a técnica **IP Checksum**, que nos permite verificar a integridade dos dados transmitidos. Esta técnica é fundamental para identificar e corrigir erros, aumentando assim a robustez e a confiabilidade do nosso sistema de comunicação.

Plataforma Arduino

Arduino é uma plataforma de prototipagem eletrónica de código aberto que é baseada em hardware e software flexíveis e fáceis de usar. O *Arduino* pode receber entradas de uma variedade de sensores e pode afetar o seu ambiente controlando luzes, motores, etc. A placa do microcontrolador na plataforma *Arduino* é programada usando a Linguagem de Programação *Arduino* (baseada em *Wiring*) e o Ambiente de Desenvolvimento *Arduino* (baseado em *Processing*). Os projetos *Arduino* podem ser autónomos ou podem comunicar com software em execução num computador. [3]

No contexto deste trabalho, o *Arduino* é usado como um emissor num **Sistema de Comunicação Digital** (**SCD**) em modo **simplex**. A comunicação é estabelecida através de uma **ligação USB** entre o *Arduino* e um computador.

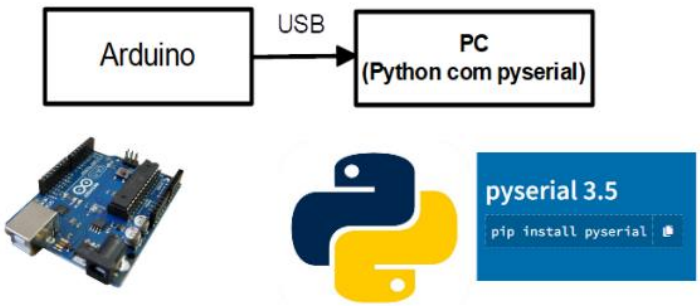

Figura 29 - Ilustração de SCD a funcionar em modo simplex com comunicação do Arduino (emissor) para o PC (recetor)

O **módulo pyserial** ilustrado na **figura 29**, é uma biblioteca *Python* que fornece acesso a funcionalidades de comunicação em série. Suporta diferentes tipos de interfaces série, incluindo **USB**, que é o que usamos neste projeto. O **pyserial** facilita a leitura e escrita de dados através da porta serial, tornando possível a comunicação entre o Arduino e o computador. [4]

No nosso trabalho, o módulo **pyserial** é usado para estabelecer uma ligação serial entre o *Arduino* (**emissor**) e o computador (**recetor**). A informação transmitida pelo *Arduino* é recebida pelo computador através desta ligação em série e é então escrita na consola.

Demonstração do bom funcionamento do SCD – Alínea (a)

Nesta primeira alínea pretende-se demonstrar o **bom funcionamento do SCD criado**, tendo em conta as especificações requisitadas. Para tal, realizou-se um programa, em C, simples, no lado do *Arduino*, que transmitisse todos os números primos entre 1 a 100, e, no lado da nossa máquina, um programa, em Pyhton, que mostra-se na consola os números obtidos da transmissão.

De forma a garantir a receção correta e ordenada da transmissão da sequência de números primos pelo *Arduino*, esta é apenas iniciada após 3 segundos da execução do programa, de forma a existir um intervalo de tempo suficiente para inicializar o recetor.

Com isto, obtiveram-se todos os números primos entre 1 a 100, confirmando o bom funcionamento do SCD criado, sem a realização de deteção de erros.

Deteção de erros em rajada – Alínea (b)

Nesta alínea pretende-se testar a deteção de erros nos bits transmitidos do *Arduino* para a nossa máquina, recorrendo à técnica de ***IP Checksum***. Para tal, recorreu-se à implementação do **canal Burst**, realizada no exercício 2, que receberá os bits transmitidos pelo *Arduino* e aplicará padrões de erro em rajada de tamanho e probabilidade definidas pelo utilizador.

Novamente, com o objetivo de garantir a receção correta e ordenada da transmissão da sequência de números primos pelo *Arduino*, esta é apenas iniciada após 3 segundos da execução do programa, de forma a existir um intervalo de tempo suficiente para inicializar o recetor.

As seguintes figuras ilustram dois exemplos de execução, onde são transmitidos apenas os números primos entre 1 e 5, repetidamente, com diferentes probabilidades de ocorrência de erros em rajada. Nestes exemplos, são mostrados os 3 números primos, enviados pelo *Arduino*, tal como o *checksum* dos mesmos, estes valores após a sua passagem pelo canal Burst e a síndroma calculada a partir destes valores.

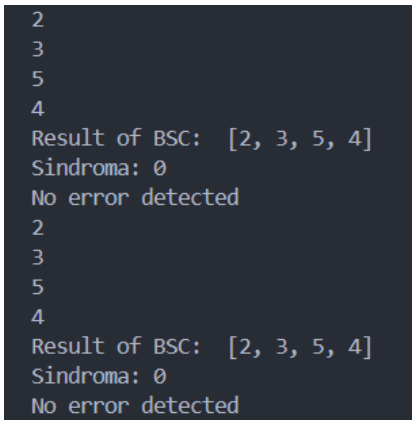

Figura 30 – Output resultante de uma transmissão com probabilidade de ocorrência de erros em rajada de tamanho 3 igual a 0,01

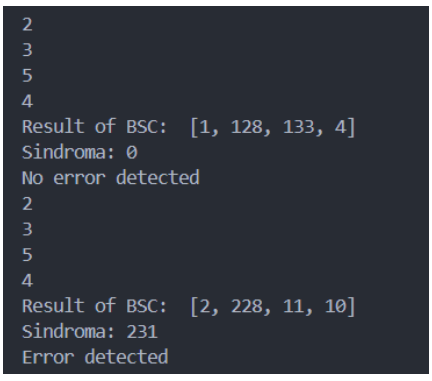

Figura 31 - Output resultante de uma transmissão com probabilidade de ocorrência de erros em rajada de tamanho 3 igual a 0,1

Com os resultados apresentados, é possível concluir que:

- Quanto maior a probabilidade de ocorrência de erros, maior será a probabilidade de a informação transmitida apresentar erros;
- Tal como para a técnica de CRC, há certas situações em que, apesar da informação transmitida apresentar erros, a técnica de *IP Checksum* não é capaz de detetar esses erros. Uma destas situações será a ocorrência de alterações simétricas nos bits da mensagem, isto é, ocorrer o mesmo número de alterações de bits a 0 para 1 e de bits a 1 para 0, resultando numa soma incapaz de detetar erros na mensagem.

Bibliografia

[1] A. Ferreira, *10. Codificação de Canal (Códigos de Controlo de Erros).*

[2] C. M. Ribeiro, Sistemas de Comunicação Digital, Lisboa: Instituto Politécnico de Lisboa, 2023.

[3] “Wikipedia,” [Online]. Available: https://pt.wikipedia.org/wiki/Arduino.

[4] C. L. Revision, “Pyseria,” [Online]. Available: https://pyserial.readthedocs.io/en/latest/.