2
Códigos LT

Os códigos LT (Luby Transform), propostos por Michael Luby em 2002 [6], foram a primeira implementação do conceito de fontana digital (*digital fountain*) com tempo de codificação e decodificação viável. As fontanas digitais são sistemas que produzem à sua saída, a partir de um bloco (ou arquivo)

$$\boldsymbol{s} = \Big(\ s_1, \ s_2, \ \ldots \ , \ s_k \ \Big)$$

com k símbolos de entrada gerados por uma fonte de informação, um bloco

$$\boldsymbol{x} = \Big(\ x_1, \ \ldots \ , \ x_j, \ \ldots \ , \ x_n \ \Big),$$

potencialmente muito grande, com n símbolos codificados.

O fato dos códigos LT se basearem em operações XOR permite a construção de codificadores e decodificadores que operam em altas velocidades [19]. Os códigos LT foram originalmente concebidos para canais com apagamento (*Erasure Channel*), livres de erros, isto é, canais em que o alfabeto de entrada é o conjunto de símbolos $\mathcal{A} = \{a_0, \ldots, a_{q-1}\}$ e o alfabeto de saída é o conjunto de símbolos $\mathcal{B} = \{a_0, \ldots, a_{q-1}, a_q\}$ e a probabilidade de um símbolo transmitido ser recebido corretamente é $(1-P_a)$ e de ser apagado é P_a.

A saída do codificador LT (entrada do canal portanto) é uma sequência de v.a.'s (variáveis aleatórias) independentes $\boldsymbol{X} = X_1, \ldots, X_k, \ldots, X_n$ e tem-se que a probabilidade condicional que relaciona a v.a. X_j à entrada do canal com a v.a. Y_j, à saída do canal, também designada por probabilidade de transição do canal, é

$$P(Y_j = a_m \mid X_j = a_i) = \begin{cases} 1 - P_a, & m = i \in \{0, q-1\} \\ P_a, & m = q \end{cases}$$

Em geral tem-se $q = 2^\ell$ e, neste trabalho, sem perda de generalidade, considera-se o caso $\ell = 1$, e que, portanto, a transmissão é feita através de um BEC (*Binary Erasure Channel*). Ainda por simplicidade, para representar o

alfabeto de entrada e saída do canal, a seguinte notação é utilizada

$$\mathcal{A} = \{a_0, a_1\} = \{0, 1\} \text{ e, } \mathcal{B} = \{a_0, a_1, a_2\} = \{0, 1, ?\}.$$

Os códigos LT são tais que cada símbolo codificado t_j é estatisticamente independente de todos os demais símbolos codificados e o conjunto de k símbolos de entrada originais pode ser recuperado, com probabilidade maior que $(1-\delta)$, a partir de quaisquer $k+O(\sqrt{k}\ln^2(k/\delta))$ símbolos codificados, com uma média de $O(k\ln(k/\delta))$ operações por símbolo [6]. O número de símbolos codificados que podem ser gerados a partir dos dados de entrada é potencialmente ilimitado. Por isso, os códigos LT são ditos *códigos de taxa versátil* [1] (*rateless*). Assim, independentemente do modelo de perdas do canal em uso, o codificador simplesmente gera e envia dados até que um número suficiente seja recebido pelo decodificador para a recuperação da mensagem original. A característica do canal somente influencia no tempo necessário até que sejam recebidos símbolos de saída em quantidade suficiente para que a decodificação seja realizada com sucesso. Em qualquer canal com apagamento, os códigos LT são quase ótimos no sentido de que o decodificador pode recuperar a mensagem original formada por k símbolos de entrada a partir de quaisquer $(1+\epsilon)k$ símbolos codificados, onde $0 < \epsilon < 1$. Considera-se um código ótimo, aquele que consegue recuperar a mensagem original (formada por k símbolos de entrada), a partir de qualquer subconjunto de k símbolos de saída dentre os n gerados pelo codificador [9]. Por serem quase ótimos e por sua eficiência aumentar com o crescimento do comprimento do bloco de entrada do codificador, os códigos LT são denominados universais [6].

O objetivo deste capítulo é o estudo dos algoritmos de codificação e decodificação, a escolha dos parâmetros da distribuição Sóliton Robusta e a análise do desempenho dessa classe de código.

2.1 Codificação

Nesta seção é descrito o processo de codificação utilizando códigos LT. Considere que a mensagem original $\boldsymbol{s}$ consiste de k símbolos de informação denotados por $s_1, s_2, ..., s_k$. Em geral, $s_i \in GF(2^\ell)$, mas por simplicidade considera-se um alfabeto de entrada binário, ou seja, $\ell = 1$. À saída do codificador um vetor $\boldsymbol{x}$ de dimensão n é obtido. É importante ressaltar que

[1] Na verdade, para um dado usuário, os códigos LT são códigos de taxa variável, sendo o número de símbolos codificados n recebidos pelo usuário variável de acordo com o número de símbolos necessários para que a decodificação ocorra com sucesso.

a regra de formação de x_n permite que o número de símbolos de $\boldsymbol{x}$, $n = |\boldsymbol{x}|$, possa ser aumentado indefinidamente de forma que a taxa deste código,

$$R = \frac{k}{n},$$

pode tender para zero.

Para descrever o processo de codificação considere inicialmente uma sequência $\{D_1, \ldots, D_n\}$ de v.a.'s discretas i.i.d.'s em que $D_j \in \mathbb{Z}_k = \{1, \ldots, k\}$. Considere também que a função μ com

$$\mu(d) = P(D_j = d), \quad d \in \mathbb{Z}_k.$$

é a Distribuição de Probabilidades (DP) associada à v.a. D_j.

Para uma dada sequência de informação $\boldsymbol{s} = (\, s_1, \ldots, s_k \,)$, considerando que $\{d_1, \ldots, d_n\}$ é uma realização da sequência aleatória $\{D_1, \ldots, D_n\}$, tem-se então que cada símbolo codificado x_j é produzido de forma aleatória de acordo com a seguinte equação

$$x_j = \sum_{\ell=1}^{d_j} s_{i_\ell(j)} \tag{2-1}$$

onde $\sum$ corresponde à adição módulo-2 dos termos do somatório.

É importante ressaltar que os valores dos índices $\{i_1(j), i_2(j), \ldots, i_{d_j}(j)\}$ das d_j parcelas $\{s_{i_1(j)}, s_{i_2(j)}, \ldots, s_{i_{d_j}(j)}\}$ em (2-1) são realizações da sequência de v.a.'s $I_1, \ldots, I_{d_j}$ onde $I_1 \in \mathbb{Z}_k$ é uma v.a. discreta com DP uniforme, ou seja, para $i_1 \in \mathbb{Z}_k$, tem-se

$$P(I_1 = i_1) = \frac{1}{k}.$$

A v.a. I_2 é também uma v.a. discreta com DP-condicional uniforme, definida da seguinte maneira para $(i_1, i_2) \in \mathbb{Z}_k^2$,

$$P(I_2 = i_2 \mid I_1 = i_1) = \begin{cases} \frac{1}{k-1} & i_2 \neq i_1 \\ 0 & i_2 = i_1 \end{cases}$$

A v.a. I_3 tem DP-condicional uniforme, definida para $(i_1, i_2, i_3) \in \mathbb{Z}_k^3$,

$$P\left(I_3 = i_3 \mid (I_1, I_2) = (i_1, i_2)\right) = \begin{cases} \frac{1}{k-2} & i_3 \notin \{i_1, i_2\} \\ 0 & i_3 \in \{i_1, i_2\}, \end{cases}$$

e, assim por diante, até que, para $(i_1, i_2, \ldots, i_{d_j}) \in \mathbb{Z}_k^{d_j}$, tem-se

$$P(I_{d_j} = i_{d_j} \mid (I_1, \ldots, I_{d_j-1}) = (i_1, \ldots, i_{d_j-1})) = \begin{cases} \frac{1}{k-(d_j-1)} & d_j \notin \{i_1, \ldots, i_{d_j-1}\} \\ 0 & d_j \in \{i_1, \ldots, i_{d_j-1}\}. \end{cases}.$$

Vale observar que $\boldsymbol{x} = (\, x_1, \ldots, x_j, \ldots, x_n \,) \in \{0,1\}^*$, onde $\{0,1\}^*$ é a notação usada para representar o conjunto de todos os blocos binários.

2.2 Codificador LT como um código em grafo

Nesta seção o codificador LT será descrito em associação com um grafo $\mathcal{G}$, bi-particionado, isto é, um grafo em que o conjunto de nós é particionado em dois conjuntos — um conjunto $\mathcal{E} = \{s_1, \ldots, s_k\}$ de *nós de mensagem* ou *nós de entrada* e um conjunto $\mathcal{S} = \{x_1, \ldots, x_n\}$ de *nós de verificação* ou *nós de saída*. A cada nó de entrada associa-se um símbolo de entrada s_i (que será identificado, indistintamente, como nó s_i), e a cada nó de saída corresponderá um símbolo codificado x_j (que será identificado como nó x_j).

Note que $\mathcal{G}$ é um grafo aleatório em que cada nó de saída x_j está conectado a d_j nós de entrada $\{s_{i_1}, \ldots, s_{i_{d_j}}\}$, onde $d_j \in \mathbb{Z}_k$ é escolhido aleatoriamente de acordo a uma DP $\mu(d)$. Tem-se assim que $\mathcal{V}_j = \{s_{i_1}, \ldots, s_{i_{d_j}}\}$ é o conjunto de nós de entrada vizinhos do nó x_j. O número de vizinhos d_j de um nó de saída x_j será referido como o *grau* do nó x_j e, a DP $\mu(d)$ será designada por *distribuição de graus* do código LT. Então, o codificador LT é descrito a seguir:

- Para cada símbolo codificado x_j, escolhe-se aleatoriamente e segundo a DP $\mu(d)$, um número d_j.
- Escolhe-se aleatoriamente, um conjunto $\{s_{i_1}, \ldots, s_{i_{d_j}}\}$, de d_j símbolos de entrada distintos (em posições $\{i_1, \ldots, i_{d_j}\}$ escolhidas de acordo com uma distribuição uniforme), como símbolos de entrada vizinhos do símbolo codificado x_j.
- O valor do símbolo codificado é

$$x_j = s_{i_1} \oplus \ldots \oplus s_{i_{d_j}},$$

onde $\oplus$ representa a adição bit a bit, modulo-2, desses d_j símbolos.

Neste processo de codificação duas distribuições de probabilidades são envolvidas: a distribuição de graus para determinar o grau de cada símbolo codificado (a distribuição *Sóliton Robusta* sendo a mais recomendada) e, outra,

distribuição uniforme para escolher os símbolos de entrada que formarão o conjunto de vizinhos do símbolo codificado.

A seguir, para descrever tal processo, apresentamos um exemplo.

Exemplo 2.1 *Considere uma mensagem formada por 5 símbolos de entrada* $s_1s_2s_3s_4s_5 = 10011$*, onde, por simplicidade, cada símbolo de entrada tem comprimento de apenas 1 "bit". Realizamos a codificação dessa mensagem gerando seis símbolos de saída* $x_1x_2x_3x_4x_5x_6$*. Assim, o processo de codificação é composto por seis etapas, uma para cada símbolo de saída gerado. A tabela (2.1) mostra o resultado do processo de codificação.*

Tabela 2.1: Processo de codificação.

símbolo	**grau** (d_j)	**vizinhos**	**valor**
x_1	2	s_1s_2	$s_1 \oplus s_2 = 1$
x_2	3	$s_1s_2s_4$	$s_1 \oplus s_2 \oplus s_4 = 0$
x_3	4	$s_1s_2s_3s_5$	$s_1 \oplus s_2 \oplus s_3 \oplus s_5 = 0$
x_4	4	$s_1s_3s_4s_5$	$s_1 \oplus s_3 \oplus s_4 \oplus s_5 = 1$
x_5	1	s_5	$s_5 = 1$
x_6	2	s_2s_5	$s_2 \oplus s_5 = 1$

O grafo resultante apresentado na Figura 2.1, ilustra o processo de codificação do Exemplo 2.1 — os símbolos codificados, x_n*, nós de saída, e suas respectivas conexões com os símbolos de entrada,* s_k*, nós de entrada, formam um grafo bipartido. Cada símbolo de saída é conectado, através de um ramo, aos símbolos de entrada vizinhos.*

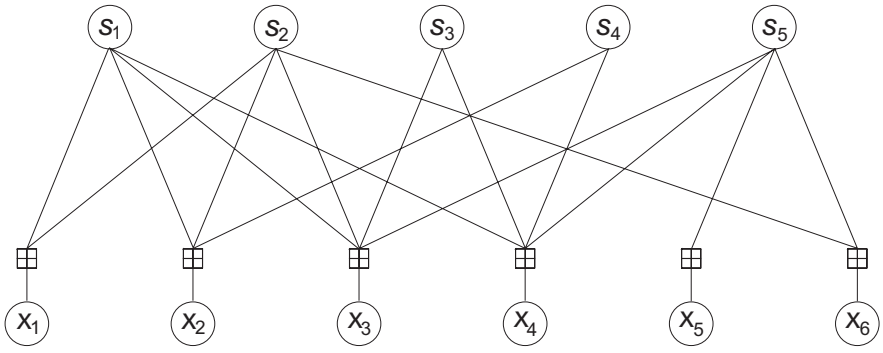

Figura 2.1: Grafo resultante da codificação.

■

2.3 Decodificação

Para recuperar os k símbolos de entrada da mensagem original, supõe-se que o decodificador conhece o grafo $\mathcal{G}$, isto é, o grau e o conjunto de vizinhos de cada símbolo codificado. Neste processo de decodificação, utiliza-se a mesma notação: os símbolos codificados (x_n) formarão os nós de saída e os símbolos de entrada (s_k) formarão os nós de entrada. O algoritmo de decodificação de códigos LT é descrito a seguir:

1. Encontrar um nó de saída x_n que esteja conectado a apenas um símbolo de entrada s_k, caso não exista tal nó de saída, o processo de decodificação falha, sendo necessário receber mais símbolos codificados antes de fazer uma nova tentativa de decodificação.

   - Fazer $s_k = x_n$,
   - Somar em modulo-2, o valor de s_k a todos os nós de saída restantes x'_n que estejam conectados a s_k,
   - Remover todas as conexões que chegam ao símbolo de entrada s_k. Como resultado, o grau de tais nós de saída é reduzido em um.

2. Repetir (1) até que todos os s_k símbolos de entrada sejam determinados.

O processo de decodificação correspondente ao Exemplo 2.1 está ilustrado na Figura 2.2. Existem cinco símbolos de entrada (indicados pelos nós de entrada superiores) que inicialmente estão apagados, e, seis símbolos de saída (indicados pelos nós de saída inferiores), os quais tem o seguinte valor inicialmente $x_1x_2x_3x_4x_5x_6 = 100111$.

1. Na primeira iteração, o único nó de saída conectado a apenas um símbolo de entrada é x_5 (Fig. 2.2-a).

2. Faz-se então $s_5 = x_5$ e elimina-se o nó de saída x_5 (Fig. 2.2-b).

3. Adiciona-se o valor de s_5 aos nós de saída que estão ligados ao mesmo, desconectando s_5 do grafo (Fig. 2.2-c).

4. No inicio da segunda iteração, o sexto nó de saída está ligado apenas a s_2 (Fig. 2.2-c).

5. Faz-se então $s_2 = x_6$ e elimina-se o nó de saída x_6 (Fig. 2.2-d).

6. Adiciona-se o valor de s_2 aos três nós de saída ligados a ele, desconectando s_2 do grafo (Fig. 2.2-e).

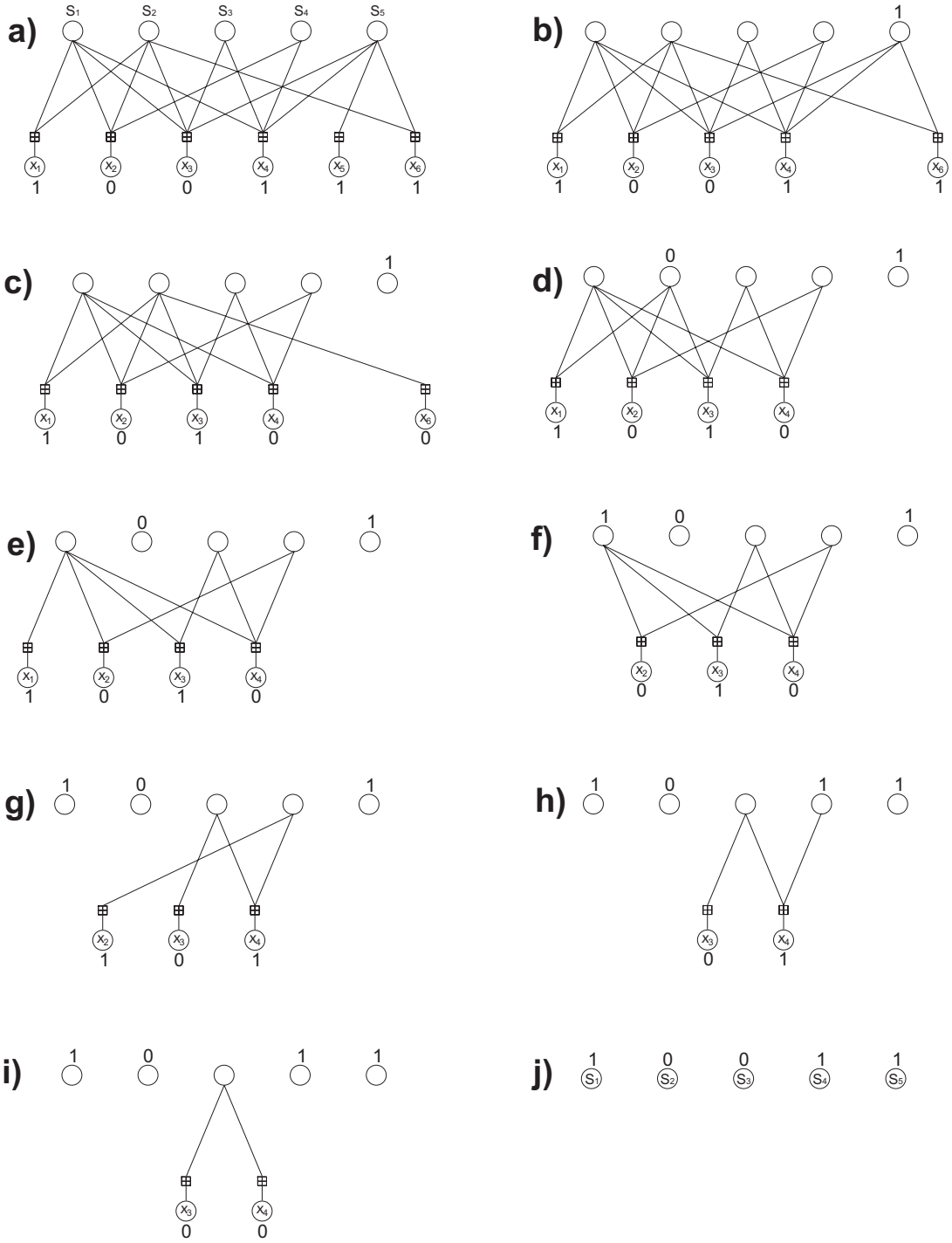

Figura 2.2: Grafo do processo de decodificação

7. No inicio da terceira iteração, o primeiro nó de saída está ligado apenas a s_1 (Fig. 2.2-e).

8. Faz-se então $s_1 = x_1$ e elimina-se o nó de saída x_1 (Fig. 2.2-f).

9. Adiciona-se o de valor s_1 aos três nós de saída ligados a ele, desconectando s_1 do grafo (Fig. 2.2-g).

10. No inicio da quarta iteração, escolhe-se o segundo nó de saída que está ligado apenas a s_4 Fig. (2.2-g).

11. Faz-se então $s_4 = x_2$ e elimina-se o nó de saída x_2 (Fig. 2.2-h).

12. Adiciona-se o valor de s_4 ao nó de saída restante ligado a ele, desconectando s_4 do grafo (Fig. 2.2-i).

13. Finalmente, percebe-se que dois nós de saída estão ligado a s_3, e ambos atribuem o mesmo valor a tal símbolo de entrada, o qual é restaurado em (j).

2.3.1 O processo LT

Introduzimos a descrição do processo LT para descrever o projeto e análise de uma boa distribuição de graus. O processo LT é uma generalização do clássico processo de lançar bolas aleatoriamente em um conjunto de urnas. Uma análise bem conhecida do processo clássico mostra que $n = k.\ln(k/\delta)$ bolas em média, são necessárias para garantir que cada uma das k urnas é coberta pelo menos por uma bola com probabilidade maior que $(1-\delta)$ [6]. Na análise do processo LT, símbolos codificados são análogos a bolas, e símbolos de entrada são análogos a urnas. O processo tem sucesso se, no final, todos os símbolos de entrada forem cobertos.

Definição 2.1 (Processo LT) *Todos os símbolos de entrada estão inicialmente descobertos — um símbolo de entrada é dito coberto se, ao longo da decodificação, recebe-se um símbolo codificado de grau unitário que o tem como vizinho. No primeiro passo do processo LT, todos os símbolos codificados com apenas um vizinho são liberados para cobrir seu vizinho único. O conjunto de símbolos de entrada cobertos que não foram ainda processados é chamado "ripple" e, portanto, neste ponto todos símbolos de entrada cobertos estão no "ripple". Em cada passo subseqüente do processo LT, um símbolo de entrada no "ripple" é processado, ou seja, ele é removido como vizinho de todos os*

símbolos codificados que o tenham como tal; e posteriormente todos os símbolos codificados que possuírem apenas um vizinho são liberados para cobrir tal vizinho restante. Alguns destes vizinhos podem ser símbolos anteriormente descobertos, fazendo que o "ripple" cresça, enquanto outros que já foram cobertos não causam alteração no "ripple". O processo termina quando o "ripple" encontrar-se vazio em algum passo. O processo falha quando existe pelo menos um símbolo de entrada não-coberto ao final do processo, e tem sucesso se no final todos os símbolos de entrada estiverem cobertos. ■

A Figura 2.3 ilustra o processo LT para a codificação do Exemplo 2.1 descrito anteriormente. Inicialmente (a) todos os símbolos de entrada (urnas) estão descobertos (urnas sem bolas). No primeiro passo todos os símbolos codificados (representados por retângulos, as bolas dentro dos retângulos representam os vizinhos do símbolo codificado) com apenas um vizinho são liberados para cobrir o mesmo (b). Em cada passo subseqüente, um símbolo de entrada do *ripple* é processado, ou seja, ele é removido como vizinho de todos os símbolos codificados que o tenham como tal, os símbolos sombreados em (c), (e), (g) e (i) indicam tal processamento. Vale mencionar que em (i) dois símbolos de entrada (s_3 e s_4) do *ripple* são processados. Em (c) o *ripple* é formado por s_5, em (e) por s_2, em (g) por s_1 e em (i) por s_3 e s_4, ou seja, o conjunto de símbolos de entrada cobertos mas ainda não processados. O processo termina quando o *ripple* encontrar-se vazio em algum passo (j). O processo é bem sucedido quando todos os símbolos de entrada são cobertos, caso contrário falha. No exemplo em questão o processo foi bem sucedido.

No processo clássico de lançamento de n bolas em um conjunto de k urnas, todas as bolas são lançadas ao mesmo tempo, causando problemas, já que várias bolas podem cobrir a mesma urna, tornando necessário um grande número de bolas para cobrir todas as urnas. Portanto, um projeto apropriado de uma distribuição de graus assegura que o processo LT libere símbolos codificados de forma incremental para cobrir todos os símbolos de entrada. Os objetivos de uma apropriada distribuição de graus são:

- Liberar lentamente símbolos codificados enquanto o processo evolui, para manter o *ripple* pequeno, de modo a evitar cobertura redundante dos símbolos de entrada no *ripple* por vários símbolos codificados.
- Não permitir que o *ripple* desapareça antes que todos os símbolos de entrada estejam devidamente cobertos.

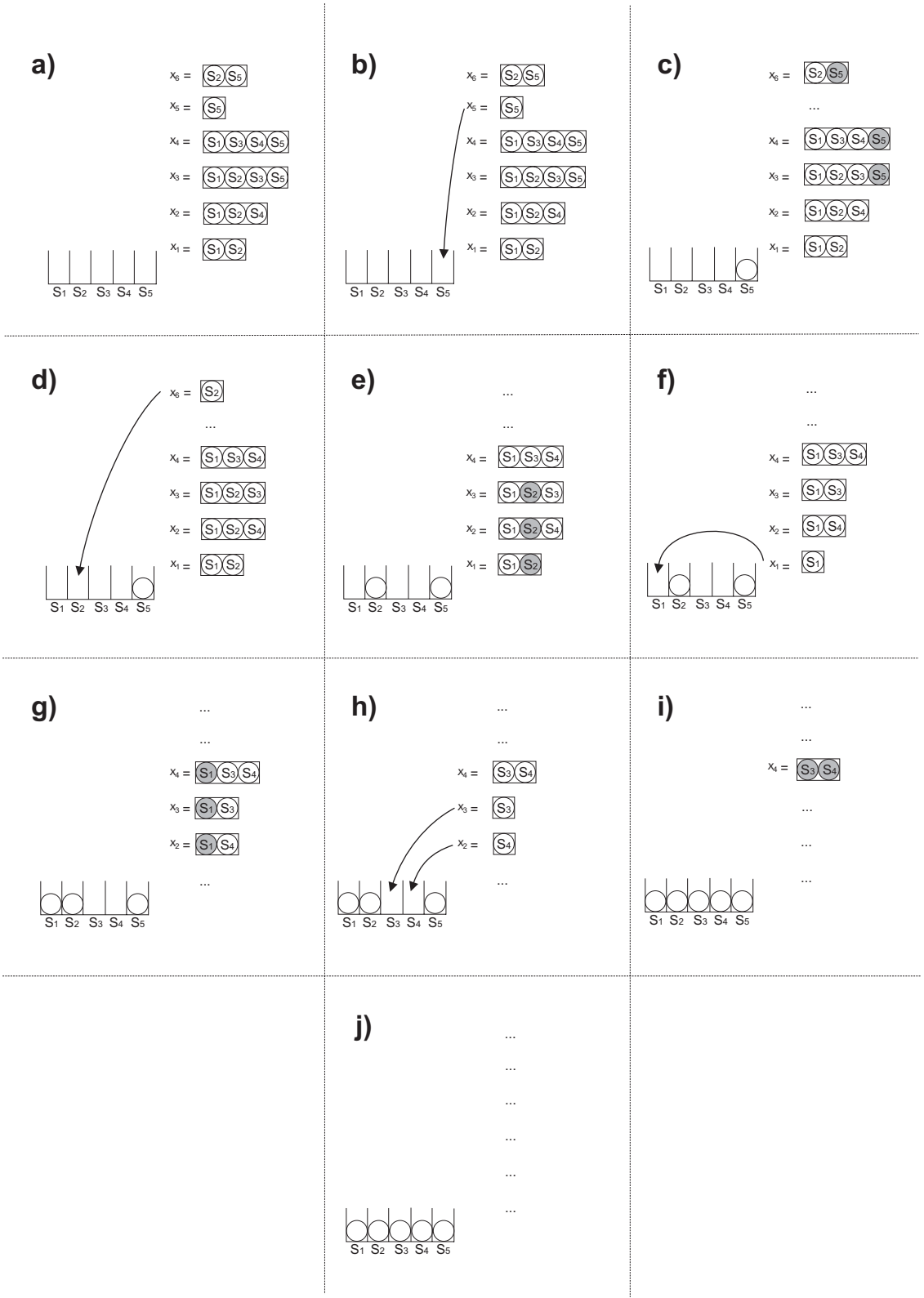

Figura 2.3: Processo LT

Não é muito difícil verificar que há uma correspondência entre o processo LT e o decodificador, isto é, um símbolo codificado cobre um símbolo de entrada se e somente se o símbolo codificado pode recuperar tal símbolo de entrada. Assim, o processo LT tem sucesso se, e somente se o decodificador consegue recuperar com sucesso todos os símbolos de entrada do arquivo original. O número total de símbolos codificados necessários para cobrir todos os símbolos de entrada no processo LT corresponde ao número total de símbolos codificados necessários para recuperar os dados de entrada no processo de decodificação. A soma dos graus dos símbolos codificados corresponde exatamente ao número total de operações necessárias para recuperar o arquivo original.

2.4 Distribuições de graus

Nesta seção, vamos introduzir a definição de duas distribuições de graus utilizadas no processo de codificação de códigos LT clássicos: a distribuição Sóliton Ideal e a distribuição Sóliton Robusta. A distribuição de grau usada para a construção de códigos LT – *Distribuição Sóliton Robusta* – é construída de tal forma que o decodificador possa recuperar a mensagem original com pouco mais que k símbolos codificados, com probabilidade no mínimo $(1-\delta)$ [6].

2.4.1 Distribuição Sóliton Ideal

A distribuição Sóliton Ideal $\rho(d)$ é definida como:

$$\rho(d) = \begin{cases} \frac{1}{k}, & \text{para } d = 1 \\ \frac{1}{d(d-1)}, & \text{para } d = 2, 3, ..., k. \end{cases} \tag{2-2}$$

A Figura 2.4 ilustra a distribuição Sóliton Ideal para uma situação em que $k = 10^4$.

2.4.2 Distribuição Sóliton Robusta

A distribuição Sóliton Robusta $\mu(d)$, associada à variável aleatória d que corresponde ao número de símbolos de entrada conectados a um dado símbolo de saída, é definida como segue.

$$\mu(d) = \frac{\rho(d) + \tau(d)}{\beta}, \quad \text{para } 1 \leq d \leq k \tag{2-3}$$

onde $\beta = \sum_{d=1}^{k} (\rho(d) + \tau(d))$ é uma constante de normalização.

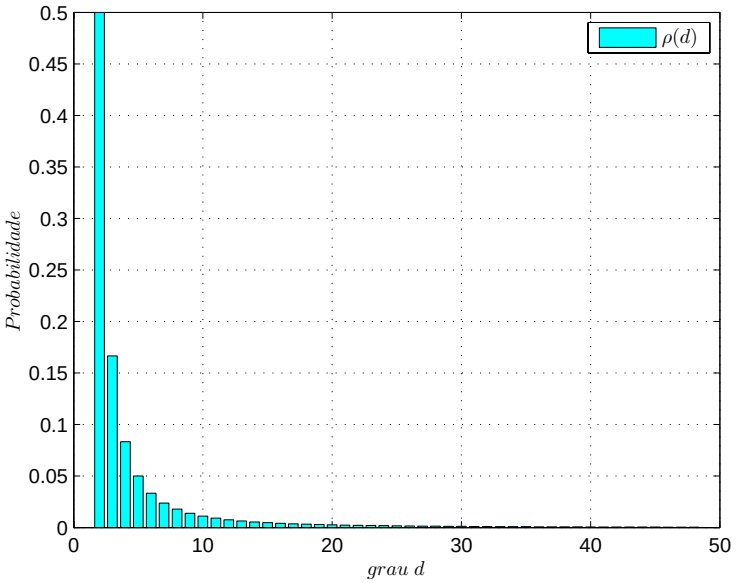

Figura 2.4: A distribuição Sóliton Ideal $\rho(d)$ para o caso $k = 10000$.

A função positiva $\tau(d)$ é dada por

$$\tau(d) = \begin{cases} \frac{\bar{R}}{dk}, & \text{para } 1 \leq d \leq \frac{k}{\bar{R}} - 1, \\ \frac{\bar{R}}{k} \ln\left(\frac{\bar{R}}{\delta}\right), & \text{para } d = \frac{k}{\bar{R}}, \\ 0, & \text{para } d = \frac{k}{\bar{R}} + 1, ..., k. \end{cases} \tag{2-4}$$

Para uma constante apropriada $c > 0$, o parâmetro $\bar{R}$ representa o número médio de símbolos codificados de grau um e é definido como

$$\bar{R} = c\sqrt{k} \ln\left(\frac{k}{\delta}\right). \tag{2-5}$$

A Figura 2.5 ilustra a função $\tau(d)$ para uma situação em que $k = 10^4$, $c = 0.2$ e $\delta = 0.1$.

Luby mostrou que, para um valor de c apropriadamente escolhido (independente de k e δ), o decodificador pode recuperar a mensagem original a partir de $n = k\beta = k + c\sqrt{k}\ln^2(k/\delta)$ símbolos codificados, com probabilidade de falha menor que δ [6].

O número de símbolos codificados é ajustado em $n = k\beta$. Isto implica que $k\left(\rho(i) + \tau(i)\right)$ é o valor esperado do número de símbolos codificados de grau i.

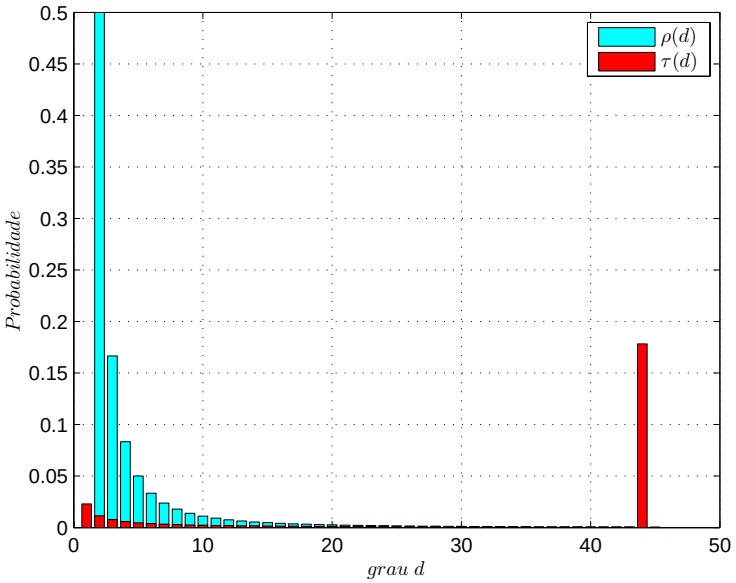

Figura 2.5: A distribuição Sóliton Ideal $\rho(d)$ e a função positiva $\tau(d)$ para o caso $k = 10^4$, $c = 0.2$ e $\delta = 0.1$.

A Figura 2.6 mostra a distribuição de grau ótima Sóliton Robusta para $k = 10^4$, $c = 0.2$ e $\delta = 0.1$. Este valor de c foi escolhido para uma melhor visualização do gráfico.

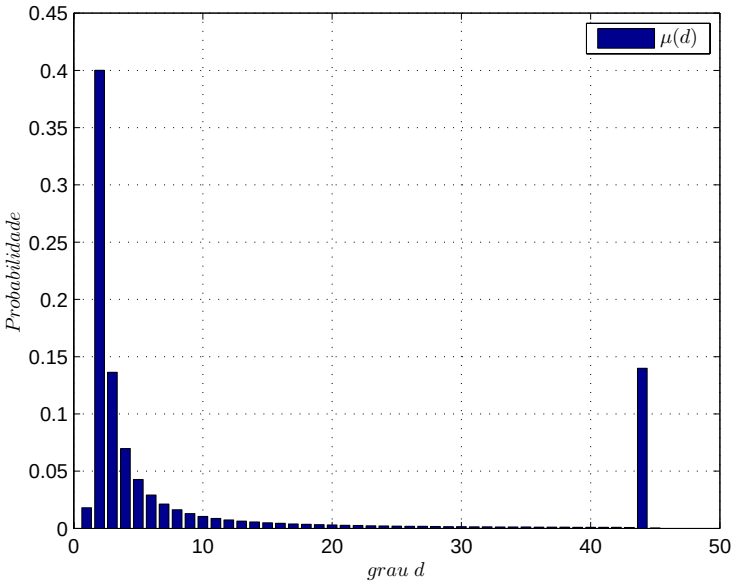

Figura 2.6: A distribuição Sóliton Robusta $\mu(d)$ para o caso $k = 10^4$, $c = 0.2$ e $\delta = 0.1$.

2.5 Método das Transformações Inversas

Esta seção apresenta um método que vai ser utilizado na implementação do codificador e do decodificador LT, mais precisamente na determinação do grau do símbolo codificado, onde se fazem uso de duas distribuições de probabilidades distintas: a distribuição Sóliton Robusta e a distribuição uniforme.

O método das transformações inversas para simular variáveis aleatórias discretas é descrito a seguir:

Para um instante, deseja-se simular uma variável aleatória discreta D a qual pode assumir k valores distintos, com distribuição de probabilidade

$$P\left(D = d_j\right) = P_j \text{ para } j = 1, \ldots, k \text{ e } \sum_j P_j = 1.$$

Então, para simular a variável aleatória discreta D, considera-se uma variável aleatória uniformemente U distribuída entre (0,1), e faz-se

$$D = \begin{cases} d_1, & \text{se } U < P_1, \\ d_2, & \text{se } P_1 \leq U < P_1 + P_2, \\ \vdots & \\ d_j, & \text{se } \sum_{i=1}^{j-1} P_i \leq U < \sum_{i=1}^{j} P_i, \\ \vdots & \end{cases} \tag{2-6}$$

Como

$$P\left(D = d_j\right) = P\left(\sum_{i=1}^{j-1} P_i \leq U < \sum_{i=1}^{j} P_i\right) = P_j,$$

pode-se concluir que D tem a distribuição de probabilidade desejada.

2.5.1 Determinação do grau do símbolo codificado

Usando o método das transformações inversas, é feita a determinação do grau (número de vizinhos) dos símbolos codificados a partir de um número uniformemente distribuído entre 0 e 1. Basta adotar $d_j = j$, onde j determina o grau do símbolo codificado e $P_j = \mu(j)$.

De acordo com a codificação de códigos LT, o primeiro passo consiste na determinação do grau do símbolo a ser codificado. Portanto, utiliza-se o método das transformações inversas para simular a distribuição Sóliton Robusta $\mu(d)$, armazenando para cada símbolo codificado um número U uniformemente distribuído entre (0,1).

2.6 Escolha dos parâmetros da distribuição Sóliton Robusta

Esta seção apresenta simulações para a escolha dos parâmetros c e δ da distribuição de graus Sóliton Robusta. Os resultados da otimização desses parâmetros são verificados comparando-os aos resultados apresentados por Mackay em [8].

Em todas as simulações, o número de símbolos necessários para decodificar com sucesso foi determinado. É fato que o número de símbolos codificados que deve ser transmitido depende da probabilidade de apagamento do canal, mas, para estes casos onde os parâmetros da distribuição Sóliton Robusta vão ser determinados, considera-se um canal sem perdas. As simulações foram implementadas considerando $k = 1000$ (número de símbolos de entrada) e variando os parâmetros c e δ.

2.6.1 Resultados obtidos das simulações

O objetivo destas simulações é verificar a influência dos parâmetros da distribuição de graus Sóliton Robusta e encontrar os parâmetros operacionais para o estudo do desempenho de códigos LT.

Primeiro, o valor operacional para a constante c foi encontrado variando tal valor, com uma probabilidade de falha constante de $\delta = 0.1$. Os resultados destas simulações são mostrados na Figura 2.7. Estes são histogramas do número de símbolos codificados que são necessários para decodificar a mensagem original com sucesso. Cada um dos histogramas mostrados foi obtido após 300 simulações para cada par de c e δ.

De acordo com os histogramas mostrados na Figura 2.7, na Tabela 2.2 encontra-se indicado o número médio de símbolos necessários para uma decodificação bem sucedida. Portanto, com estes resultados obtidos, é claro que um valor operacional para a constante c é 0.03, já que com esse valor consegue-

se em média um menor número de símbolos necessários para a decodificação e uma boa distribuição de tais valores.

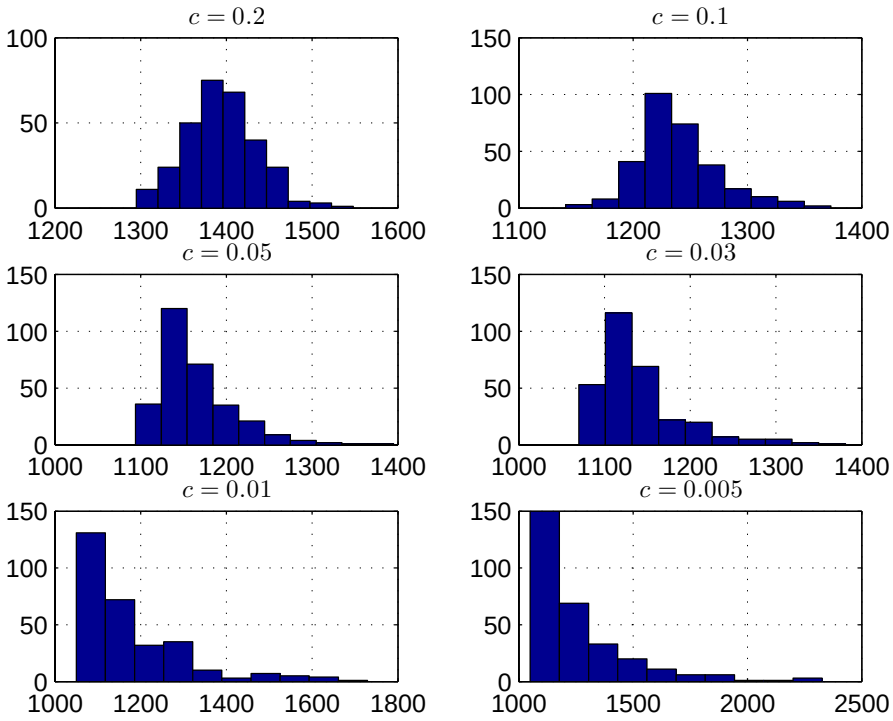

Figura 2.7: Histograma do número de símbolos codificados necessários para a decodificação com sucesso para distintos valores de c ($k = 1000$, $\delta = 0.1$).

Tabela 2.2: Número médio de símbolos necessários à decodificação bem sucedida para distintos valores de c ($k = 1000$, $\delta = 0.1$).

$\delta = 0.1$						
c	0.2	0.1	0.05	0.03	0.01	0.005
média	1394	1240	1164	1140	1176	1255

Em seguida, com este valor operacional de $c = 0.03$ encontrado novas simulações foram realizadas para encontrar o valor operacional de δ. O resultado de tal procedimento é mostrado na Figura 2.8 e na Tabela 2.3 onde encontra-se indicado o número médio de símbolos necessários à decodificação bem sucedida para distintos valores de δ quando $c = 0.03$. Pode-se observar que, para valores de δ igual a 0.5, 0.3, 0.2 e 0.1, o número médio de símbolos necessários para a decodificação é quase o mesmo. Portanto, é possível usar quaisquer desses valores de δ anteriormente mencionados. É importante observar e comentar

que à análise probabilística para determinar δ como limitante superior da probabilidade de falha na decodificação é muito pessimista, já que a probabilidade de falha real, geralmente, é muito menor que δ [6, 15]. Por isso, códigos LT podem ser projetados com grandes valores de δ e ainda apresentam bom desempenho. Destas simulações, pode-se também observar que o parâmetro c tem uma influência muito maior no desempenho e a obtenção operacional de c é importante.

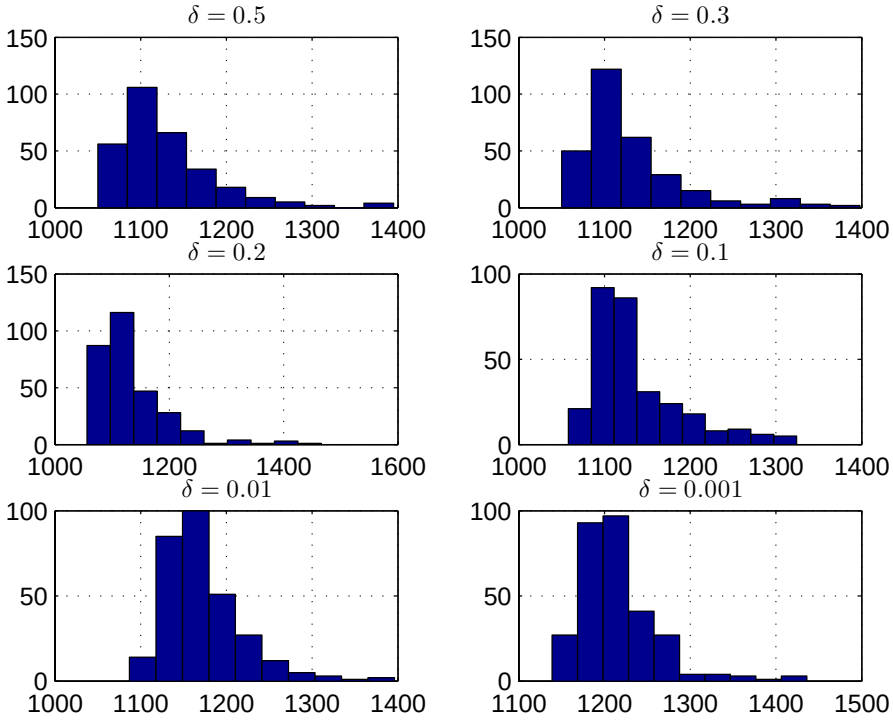

Figura 2.8: Histograma do número de símbolos codificados necessários para a decodificação com sucesso para distintos valores de δ ($k = 1000$, $c = 0.03$).

Tabela 2.3: Número médio de símbolos necessários à decodificação bem sucedida para distintos valores de δ ($k = 1000$, $c = 0.03$).

$c = 0.03$						
δ	0.5	0.3	0.2	0.1	0.01	0.001
média	1129	1130	1133	1137	1173	1215

Ao longo da dissertação, pacotes de tamanho $k = 1000$ serão usados e os códigos LT utilizados farão uso da distribuição Sóliton Robusta com parâmetros c e δ iguais a 0.03 e 0.1 respectivamente, tendo em vista que se

verificou, ao longo das simulações anteriores, que estes valores geram bons códigos LT. Considera-se um bom código LT, aquele que é capaz de recuperar a mensagem original a partir de um número de símbolos de saída n que é um pouco maior ($\approx 10\%$) que o número de símbolos de entrada k, que formam a mensagem original (Para valores maiores de k, este percentual é menor).

2.7 Análise de desempenho

Nos códigos LT, a probabilidade de sucesso na decodificação somente é limitada se o número de símbolos de saída disponíveis ao receptor for limitado. O que pode ocorrer é um maior tempo de espera para a decodificação completa da mensagem original (bloco de k símbolos de entrada). Para garantir o sucesso na decodificação, pelo menos $n = k\beta$ símbolos codificados devem ser recebidos.

Nesta seção é analisado o desempenho de códigos LT construídos com uma distribuição Sóliton Robusta que como será justificado no Capítulo 3 é mais apropriada do que a distribuição Sóliton Ideal. O estudo de códigos LT em canais com apagamento avalia o desempenho deste, tomando-se em consideração as características do canal.

2.7.1 Desempenho em canais ideais

É analisado aqui o número de símbolos codificados necessários para a decodificação de códigos LT em canais ideais (livres de perturbações). As simulações foram feitas para diferentes valores de k (número de símbolos de entrada). Ao longo do processo de decodificação, caso não se obtivesse sucesso, o receptor requisitava mais símbolos de saída ao transmissor. Tal procedimento foi realizado com a finalidade de obter uma aproximação real do número de símbolos necessários à decodificação bem sucedida. No caso de uma implementação prática, não existe necessidade de requisitar ao transmissor mais símbolos de saída, o decodificador começa a decodificação assim que estima possuir símbolos suficientes para uma decodificação com sucesso. Entretanto, só deixa de aceitar símbolos de saída provenientes do transmissor quando a mensagem original for recuperada, eliminando assim a necessidade de um canal reverso para requisitar mais símbolos de saída.

Em [14], aconselha-se que o número de símbolos extra requisitados ao transmissor (G), quando ocorre uma falha na decodificação, seja $G = \sqrt{k}$. Nas simulações aqui realizadas, escolhe-se o valor $G = 1$ a fim de obter uma maior

precisão a respeito do número real de símbolos necessários à decodificação bem sucedida. Na Tabela 2.4, encontram-se indicados o número médio de símbolos necessários à decodificação bem sucedida, para cada valor de k em função dos parâmetros $c = 0.03$ e $\delta = 0.1$.

Tabela 2.4: Número médio de símbolos necessários à decodificação bem sucedida para distintos valores de k ($c = 0.03$, $\delta = 0.1$)

$c = 0.03$ e $\delta = 0.1$					
k	100	500	1000	2000	3000
média	137	594	1130	2204	3255

Cada uma das médias obtidas foi calculada após 250 simulações para cada valor de k. O número de simulações foi escolhido de maneira arbitrária, tendo em vista que, o número de símbolos de saída necessários para a recuperação da mensagem original, mostrou-se mais concentrado em torno do valor médio exposto na Tabela 2.4.

Finalmente, na Figura 2.9 mostramos que o desempenho dos códigos LT melhora com o crescimento do tamanho do arquivo original, ou seja, a relação entre o número de símbolos de saída e de entrada (*overhead*) é menor conforme o número de símbolos de entrada k que formam o arquivo original a ser transmitido vai aumentando.

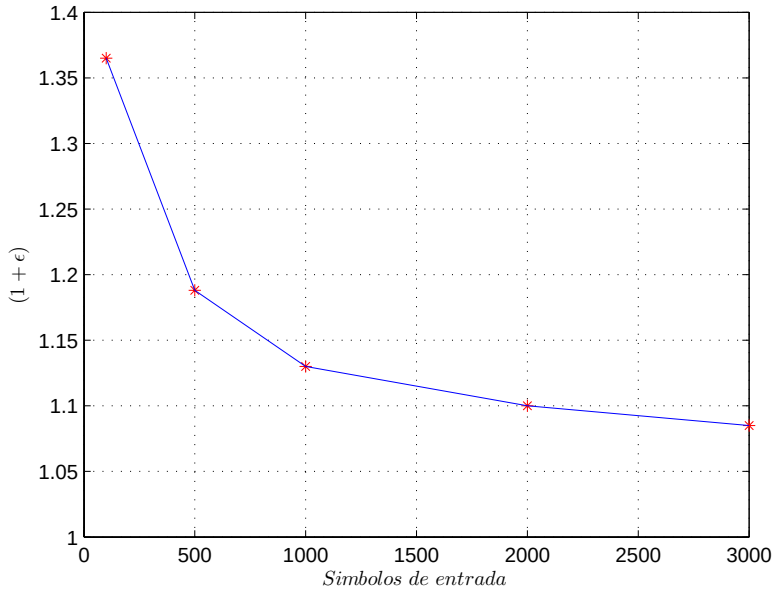

Figura 2.9: Desempenho do código LT em canais ideais.

2.7.2 Desempenho em canais com apagamento

O estudo do desempenho em canais com apagamento é importante, já que, como visto anteriormente, um canal que pode corromper ou perder pacotes pode ser modelado como um canal com apagamento, quando se trata da transmissão de pacotes em uma rede de comunicação. Para isso, assume-se que códigos detetores de erros (usados independentemente dos códigos LT) são usados para detectar erros em um pacote, conforme ilustrado na Figura 2.10. Caso um dado pacote contenha erros, o código detector de erros é usado para identificar o pacote que contém erros, tratando os mesmos como apagamentos. Logo, os apagamentos são descartados pelo decodificador LT. Assim, tem-se unicamente duas possibilidades: o pacote é recebido completamente sem erros ou é descartado.

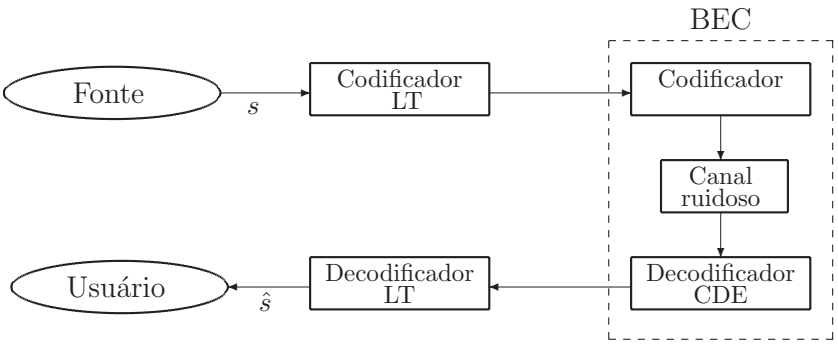

Figura 2.10: Sistema com códigos detetores de erros (CDE) usados em conjunto com códigos LT.

A Figura 2.11 mostra a quantidade de símbolos necessários para uma decodificação com sucesso, em função da probabilidade de apagamento do canal. Pode-se perceber que, à medida que a probabilidade de apagamento cresce, maior é a quantidade de símbolos codificados necessários para una decodificação bem sucedida. Isto é, quanto maior é a probabilidade de apagamento, menos símbolos de saída chegam ao decodificador. É possível observar ademais que, para probabilidades de apagamento muito baixas, o número de símbolos codificados necessários para a decodificação bem sucedida tende a ser uma constante igual a taxa mínima do código LT em uso, ou seja, o número mínimo de símbolos de saída necessários para a decodificação. Isso acontece porque praticamente todos os pacotes enviados são recebidos pelo decodificador.

Para o caso considerado neste trabalho, cada símbolo é representado por um *bit*, então o canal reduz-se ao canal binário com apagamento (BEC).

As simulações foram feitas considerando $k = 1000$. O valor $(1 + \epsilon)k = \beta k$ representa o número médio de símbolos de saída necessários para uma decodificação bem sucedida.

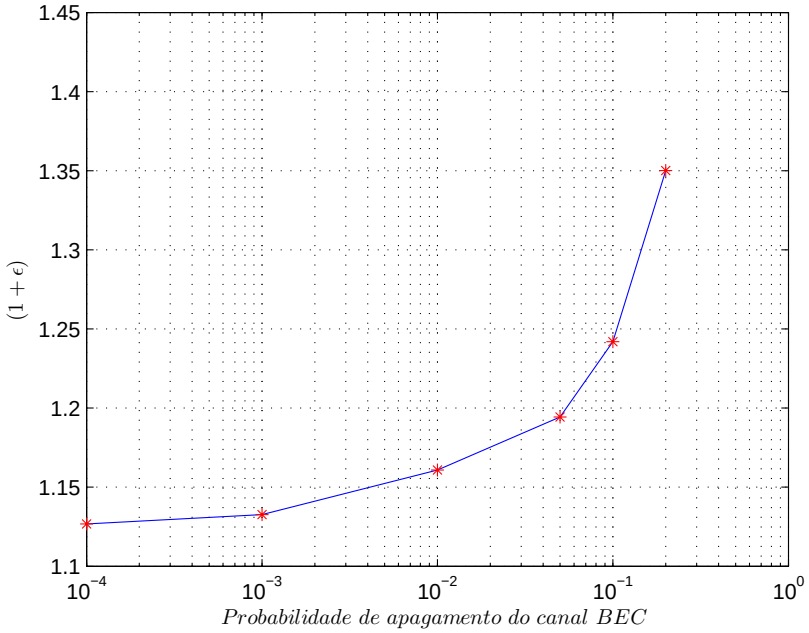

Figura 2.11: Desempenho do código LT em canais com apagamento.

2.8 Comparação com códigos tradicionais para canais com apagamento

Tipicamente, os códigos tradicionais para canais com apagamento são códigos de bloco com taxa fixa, isto é, k símbolos de entrada são usados para gerar $(n - k)$ símbolos redundantes para um total de n símbolos codificados, e a taxa do código é $R = k/n$. Têm-se sugerido o uso de códigos Reed-Solomon [16] e códigos Tornado [27] para aplicações de distribuição de dados de maneira confiável onde são requeridos códigos para canais com apagamento. Entretanto, existem severas limitações para implementar tais códigos, por exemplo k e n são limitados a valores razoavelmente pequenos ou são fixados antes que o processo de codificação comece, sendo assim, melhor a utilização de códigos LT. Entre as principais vantagens dos códigos LT em relação aos códigos tradicionais, destacam-se:

- Tempo de codificação e decodificação da ordem de $k \ln(k/\delta)$.
- Capacidade de gerar um número ilimitado de símbolos codificados.

- Geração de símbolos de saída em tempo real.
- Não é necessário conhecer *a priori* a característica do canal.

2.8.1 Códigos Reed-Solomon

Os códigos Reed-Solomon são os melhores conhecidos como códigos MDS (*Maximum Distance Separable*). Uma grande vantagem dos códigos Reed-Solomon é que, sendo a mensagem original formada por k símbolos de entrada, quaisquer k dos n símbolos codificados gerados são suficientes para recuperar a mensagem original. Por outro lado, a desvantagem destes códigos é a limitação a valores pequenos do número de símbolos de entrada k e número de símbolos de saída n, devido a considerações práticas como o tempo de codificação e decodificação. No entanto, para estes códigos, a ordem do Corpo de Galois em uso limita o número de símbolos de saída que podem ser gerados. Em implementações práticas, um valor típico para a ordem do Corpo de Galois é 256, o qual limita n a 256 símbolos.

Essa limitação no número de símbolos que podem ser gerados, fazem que os códigos Reed-Solomon não sejam apropriados para algumas aplicações como, por exemplo, distribuição de dados em grande quantidade sobre a Internet. Para um valor baixo de n, pode acontecer de não serem recebidos os k símbolos necessários à decodificação. Nesse caso, o receptor terá que esperar um outro ciclo de transmissão, recebendo um alto número de símbolos duplicados, diminuindo a eficiência.

Na prática, os códigos Reed-Solomon são somente eficientes para valores pequenos de k e n. Isto deve-se a que, para uma implementação padrão destes códigos, $k(n-k)A/2$ operações por símbolo são necessárias para produzir os n símbolos codificados, onde A é o tamanho do corpo finito usado [2] [16].

2.8.2 Códigos Tornado

Os códigos Tornado [26] são códigos de bloco baseados em grafos esparsos irregulares. Estes códigos podem ser projetados usando um alfabeto de tamanho arbitrário.

A vantagem dos códigos Tornado é que apresentam tempos de codificação e decodificação que variam linearmente com o tamanho do arquivo a ser codificado [2] [26]. Tais códigos se assemelham aos códigos LT, já que usam

uma regra similar na decodificação para recuperar os dados originais, isto é, $(1+\epsilon)k$ símbolos de saída devem ser recebidos para garantir uma decodificação com sucesso. Algumas das limitações destes códigos em relação aos códigos LT são:

- Apresenta complexidades em sua estrutura. Os códigos Tornado utilizam uma sequência cascateada de grafos bipartidos entre muitas camadas de símbolos, onde os símbolos de entrada encontram-se na primeira camada e os símbolos redundantes encontram-se em cada camada subseqüente. Na prática, é requerida a mesma estrutura gráfica para ambos, codificador e decodificador, ou uma estrutura gráfica no codificador que logo é comunicada ao decodificador. No entanto, este pré-processamento mostra-se muito dispendioso na prática, sendo o tamanho da estrutura dos grafos proporcional ao número de símbolos de saída n, necessitando-se de diferentes estruturas de grafos para cada tamanho de arquivo. Ao contrário, com o uso de códigos de LT, o único pré-processamento necessário é o cálculo das distribuições de graus usadas na codificação.
- Impossibilidade de gerar mais símbolos codificados em tempo real, caso não sejam recebidos os $(1+\epsilon)k$ símbolos necessários na decodificação, já que k e n são fixados antes do processo de codificação. Pelo contrário, o codificador LT pode gerar símbolos em tempo real até conseguir que a decodificação seja bem sucedida.
- Tem-se que conhecer *a priori* a característica do canal em uso, já que, assim como os códigos Reed-Solomon, os códigos Tornado devem ter uma taxa pré-fixada. Assim, o que se faz na prática é usar a maior probabilidade de apagamento possível para garantir o sucesso na decodificação.

Os códigos Reed-Solomon e Tornado são códigos sistemáticos, isto é, todos os símbolos de entrada que formam o arquivo original aparecem explicitamente no bloco de símbolos da sequência codificada, enquanto que códigos LT em geral são não-sistemáticos [6].