Universidade Federal do Rio de Janeiro
Escola Politécnica
Departamento de Eletrônica e de Computação

Codificação de canal com taxa variável

Autor: _____
Ricardo Dória Loyola-Camorim

Orientador: _____
Prof. Sérgio Lima Netto, Ph. D.

Examinador: _____
Prof. Eduardo Antônio Barros da Silva, Ph. D.

Examinador: _____
Prof. Gelson Vieira Mendonça, Ph. D.

DEL

Agosto de 2010

UNIVERSIDADE FEDERAL DO RIO DE JANEIRO
Escola Politécnica – Departamento de Eletrônica e de Computação
Centro de Tecnologia, bloco H, sala H-217, Cidade Universitária
Rio de Janeiro – RJ CEP 21949-900

DEDICATÓRIA

Dedico esse trabalho à minha família, que sempre foi a base para tudo o que eu construí na vida. Em especial, dedico ao meu pai e à minha mãe, que me deram tudo que eu precisei e muito mais.

AGRADECIMENTO

Agradeço à minha família que me apoiou sempre, cada um de sua forma. Em especial, minha mãe, meu pai, meus irmãos e minha avó.

Sou grato, também, aos professores que participaram de minha formação, desde o Colégio de São Bento até o Departamento de Eletrônica da UFRJ. Em especial, agradeço ao meu orientador, professor Sérgio Lima Netto, meu orientador acadêmico, professor Jomar Gozzi e o coordenador do curso de Engenharia Eletrônica e de Computação, professor Carlos José Ribas D'Avila. Eles desempenharam seus papéis de forma exemplar sempre me ajudando quando necessário.

Agradeço, por fim, aos meus amigos e companheiros que, de alguma forma, contribuíram para que esse trabalho se tornasse possível.

RESUMO

O projeto consiste na avaliação de uma proposta de distribuição probabilística a ser usada na codificação LT. Para que a análise da proposta seja feita, diversas outras fases são desenvolvidas.

O trabalho começa com uma fundamentação teórica onde são expostos os conhecimentos de Teoria da Informação necessários para a compreensão do projeto. Em seguida, em Matlab, são implementados o codificador, o decodificador e o simulador de canal. O codificador utiliza o algoritmo LT, que é englobado nos códigos Fontanais. Ele possui uma distribuição probabilística como entrada. Após isso, são testadas distribuições tradicionalmente utilizadas. Então, a nova proposta é testada. Por fim, todos os resultados atingidos são comparados, objetivando sua avaliação

Palavras-Chave: Código LT; Códigos Fontanais; Taxa variável; Codificação de Canal; Teoria da Informação

ABSTRACT

This project involves the evaluation of a proposal for a probabilistic distribution to be used in an LT code. In order to accomplish the analysis, some phases must also be developed.

The work begins with a theoretical framework where the necessary knowledge about Information Theory is explained for the best comprehension of the project. Afterwards, using Matlab, the encoder, the decoder and the channel simulator are implemented. The encoder uses the LT algorithm, which is encompassed by the Fountain Codes. It has a probability distribution as an input. After that, the traditionally used distributions are tested. Then, the new proposal is tested. Finally, all the results achieved are compared, aiming the distributions evaluation.

Key-words: LT Codes; Fountain Codes; Rateless Codes; Channel Coding; Information Theory

SIGLAS

UFRJ – Universidade Federal do Rio de Janeiro

VA – Variável Aleatória

XOR – Exclusive Or (Ou Exclusivo)

TCP – Transmission Control Protocol (Protocolo de Controle de Transmissão)

LT – Luby Transform (Transformada de Luby)

PDF – Probability Density Function (Função de densidade de probabilidade)

Sumário

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

1.1 – Tema

O tema deste projeto de graduação é, em um escopo mais geral, Teoria da Informação. Mais especificamente, trata-se da área de transmissão de dados, onde são aplicados códigos de canal para detecção e possível correção de erros. Será estudado e implementado o algoritmo de Luby. Este faz com que a taxa na codificação seja variável.

Os códigos que utilizam esse algoritmo têm como entrada, além da mensagem a ser transmitida, uma distribuição estatística. Será proposta, nesse projeto, uma distribuição diferente daquelas mais tradicionalmente utilizadas. Com essa mudança, deseja-se aumentar a eficiência do código resultante. Ou seja, enviar mais bits da mensagem original, utilizando menos símbolos.

1.2 – Delimitação

O trabalho realizado é relativo a quaisquer transmissores de dados cujos meios de transmissão possam ser modelados como canais de apagamento. Um exemplo típico desse tipo de canal são as redes de internet. Por exemplo, o envio de informação via web pode utilizar a codificação exposta nesse trabalho.

Além disso, os testes realizados podem servir de base para outras análises. Afinal, no trabalho são feitas diversas análises que podem ser úteis no contexto de desenvolvimento de novos códigos.

Por fim, esse projeto pode ter continuação, aprofundando-se o estudo das novas distribuições propostas.

1.3 – Justificativa

O objetivo principal de um sistema de comunicação é transmitir uma mensagem através de um canal. Ou seja, fazer com que o receptor seja capaz de determinar o conteúdo enviado com confiabilidade, mesmo diante de adversidades impostas pelo meio de transmissão. Entretanto, esse não é o único fator determinante para transmissões de dados. É necessário, também, que isso aconteça com a maior velocidade de transmissão possível. Para que isso seja possível, são usados códigos de canal. Eles transformam a mensagem de forma que, mesmo que ocorram erros, o receptor consiga recebê-la corretamente.

Em geral, são utilizados códigos de taxa fixa. Nesses casos, os meios de transmissão têm uma qualidade que não se altera muito. Logo, pode-se estimar a melhor taxa que pode ser alcançada, para um determinado código.

Contudo, ao analisarmos canais cuja taxa de erros oscile bastante, muda-se o panorama. Utilizando-se uma codificação dimensionada para certa taxa fixa, há momentos nos quais o rendimento fica abaixo do que poderia ser alcançado. Outrossim, ocorrem erros quando a qualidade do canal fica abaixo do estimado.

Justifica-se, portanto, o desenvolvimento de um código de taxa variável. Seu objetivo é alterar o número de símbolos transmitidos, dependendo das necessidades impostas pelo canal. Dessa forma, maximiza-se o tempo de utilização da menor taxa possível. É mantido, portanto, o rendimento máximo. Caso a qualidade do canal comece a diminuir e alguns dados sejam comprometidos, a taxa é aumentada. Contudo, ela não cresce de forma descontrolada. Assim sendo, a transmissão permanece sendo tão eficiente quanto o possível.

Dentro do escopo de codificações com taxa variável, há um tipo de código que é denominado código LT, o qual utiliza distribuições estatísticas. Diversas delas já foram implementadas e testadas. Contudo, visando melhoras, é importante fazer experimentos em novas propostas.

1.4 – Objetivos

Os objetivos do projeto, em termos gerais, são dois: a implementação de um codificador que use o algoritmo de Luby e a avaliação de novas propostas de distribuição estatística usada nesse codificador.

Mais explicitamente, os objetivos são os seguintes:

- A implementação de um código de canal com taxa variável utilizando o algoritmo de Luby.

- O teste de distribuições estatísticas tradicionalmente usadas como entrada do codificador.

- O teste de novas propostas de distribuições estatísticas usadas como entrada do codificador.

- A comparação dos resultados dos testes feitos no projeto.

- A avaliação da nova proposta de distribuição estatística.

1.5 – Metodologia

O projeto começa com a implementação de um exemplo simples de código com taxa variável. Utilizando Matlab, é criado um codificador que envia as mensagens e um decodificador para recebê-las. A codificação, nessa primeira parte, é fixa e independe de uma distribuição estatística.

É desenvolvido, também em Matlab, um simulador de canal com apagamento. Ele consistirá, simplesmente, de um gerador aleatório de falhas. Dada uma mensagem de entrada, apenas um percentual fixo dessa será recebido do outro lado.

Na segunda parte, então, ocorrerá a generalização do exemplo. Ou seja, a implementação do algoritmo de Luby, o qual depende de uma árvore de distribuição

estatística. Após haver desenvolvido o programa, serão testadas algumas distribuições estatísticas já analisadas em outros artigos. Os testes serão feitos variando-se o tamanho da mensagem e a taxa de falhas do canal.

A terceira e última parte consistirá na proposição de novas distribuições de entrada para a codificação. Elas serão testadas da mesma forma que as anteriores. Com o desempenho de todos os casos tendo sido registrados, os resultados do uso de cada distribuição estatística serão comparados. Essa comparação tem por objetivo a avaliação do desempenho das novas propostas.

1.6 – Descrição

No capítulo 2 é feita uma fundamentação teórica. Nele são expostos os fatores de Teoria da Informação necessários para a correta compreensão do restante do projeto. Ele começa com um histórico para situar o leitor. Então, fala-se sobre entropia e algumas definições advindas dela. O capítulo segue com sistemas de comunicação, codificação de canal, falando-se de Hamming, por exemplo, e capacidade de canal. Para finalizar, então, é discutido o canal com apagamento.

O capítulo 3 apresenta a codificação usando o conceito de códigos fontanais. Fala-se, então, do principal exemplo que são os chamados "Códigos LT". Após mostrar como são a codificação e a decodificação, mostram-se sua implementação e a metodologia de testes. Por fim, são testadas algumas distribuições de entrada para avaliar seus desempenhos.

No capítulo 4, as novas propostas são apresentadas. Primeiramente dá-se a motivação das novas proposições. Em seguida, os testes de desempenho são feitos. Por fim, os resultados são comparados com os equivalentes do capítulo 3.

Finalmente, no capítulo 5 ocorre a conclusão do projeto. É verificado o atendimento dos objetivos e, por fim, são feitas propostas de melhorias futuras.

Capítulo 2

Fundamentação Teórica

2.1 – Histórico

Atualmente, a Internet e os telefones celulares são parte do cotidiano de quase todo mundo. Além disso, a televisão digital deve, rapidamente, se popularizar. Contudo, poucos sabem que essas tecnologias dependem, fortemente, do trabalho e das descobertas de um engenheiro e matemático nas décadas de quarenta e cinquenta. Ele trabalhava na Bell Thelephone Laboratories e seu nome era Claude Elwood Shannon (1916-2001) [1].

Desde quando as telecomunicações começaram a utilizar sistemas digitais, preocupa-se com os erros que ocorrem ao se transmitir dados. É natural pensar que, para uma mesma forma de transmissão, quanto maior a taxa de dados enviados, maior é a probabilidade de erro. Afinal, uma taxa maior implica em menos tempo para cada ficar bit ativo no canal. Logo, o ruído introduzido pelo meio altera a informação mais facilmente. Entretanto, em 1948, Shannon publica o seu artigo mais conhecido: "A Mathematical Theory of Communications" [2]. Nele, é demonstrado que o aumento de taxa não implica, necessariamente, em um aumento de probabilidade de erro.

Shannon começa explicitando o problema principal em comunicação: Reproduzir em um ponto, uma mensagem escolhida em outro ponto [1]. Esta pode ser uma carta, uma foto, um bit ou qualquer outro dado que se deseje transmitir. A partir disso, ele desenvolve uma teoria matemática para estudar os meios de transmissão, a sua velocidade, etc. Essa teoria abrangia todos os canais. Desde telefones até televisão e radares.

Com essa generalização, as telecomunicações foram levadas para o campo da matemática. A informação poderia ser medida em termos de "binary digits" (dígitos binários), de onde se origina o termo 'bits' [1]. "Shannon foi a pessoa que viu que o

dígito binário era o elemento fundamental em todas as telecomunicações. Foi essa, na verdade, sua descoberta, e, dela, a revolução nas comunicações surgiu." segundo Dr. Robert G. Gallager, um colega de Shannon no Massachusetts Institute of Technology [3].

Em seu artigo, Shannon demonstra que há um valor, determinado **Capacidade de Canal**, o qual delimita a taxa de transmissão sem erros. Ele é dado em bits por segundo. Esse limite é muito importante no campo das telecomunicações. É como a velocidade da luz na física moderna ou a Segunda Lei da Termodinâmica [4]. É demonstrado que qualquer informação transmitida a uma taxa menor do que a capacidade de canal pode ocorrer com um percentual de erro tão pequeno quanto se queira. Em contrapartida, caso se enviem dados a uma taxa maior, parte da informação, certamente, irá se perder.

A perspectiva de transmissão sem erros não era considerada, na época. Afinal, sempre há ruído no canal. O modo de superá-lo era aumentar o tempo de representação do bit, o que diminuía a taxa de transferência, ou aumentar a energia do sinal. Shannon, então, introduz um método melhor de esquivar-se do problema: **Codificação.**

O trabalho de Shannon não foi limitado somente à área de transmissão de dados. É importante ressaltar, também, que seu artigo teve impacto em diversos outros campos. Tais como: compressão de dados, criptografia e inteligência artificial.

Nesse capítulo, serão vistas as conclusões de Shannon relativas à transmissão de dados. Começa-se falando sobre entropia e outras grandezas relacionadas a ela. Vêem-se suas propriedades e relações. Em seguida, são abordados os sistemas de comunicação. Ele é modelado e suas partes são explicadas. Após isso, fala-se sobre códigos de canal. São dados diversos exemplos e eles são analisados. Então, é exposta a capacidade de canal. São vistas, também, suas propriedades. Para finalizar o capítulo, é visto o canal com apagamento. Sua análise é muito importante, pois esse é o tipo de canal que é utilizado nesse trabalho.

2.2 – Entropia

A informação em uma mensagem pode ser quantificada em termos de bits. Mas como é possível fazer essa medida? Como quantificar palavras, imagens, sons, ou qualquer outra informação em termos de bits? Não se deve apenas transformar o dado em uma palavra binária. Assim, provavelmente, seriam inseridos bits a mais do que o necessário. Além disso, como será visto a seguir, a medida de informação pode ser fracionária. Dessa forma, não há uma palavra binária equivalente ao dado. Para se medir a informação, então, define-se a **Entropia**. Esta grandeza mede a incerteza de uma determinada VA.

Sabe-se que há uma relação entre a incerteza e a informação. Podemos exemplificar isso com um caso extremo. Supondo que uma mensagem contenha apenas '0'. Ou seja, a probabilidade de um bit ser '0' é 100%. Então não precisamos verificar qual o valor de cada um deles. Já se sabe que ele tem valor '0'. Em outras palavras, nenhuma informação é carregada. Assim sendo, vemos que quanto menor a incerteza, menor será a quantidade de informação. De forma análoga, se for nula a probabilidade do bit ser '0', ele também não carregará nenhuma informação. Em contrapartida, se a probabilidade de ele ser '0' ou '1' forem iguais, a incerteza é a máxima possível. Dessa forma, a quantidade de informação contida em cada bit será máxima.

A entropia é, por fim, a grandeza que define o menor número de bits necessários para se escrever uma determinada VA. Ela é definida, matematicamente, da seguinte forma:

$$H(X) = -E_p\left[\log_2 p(x)\right] = -\sum_{x \epsilon X} p(x).\left[\log_2 p(x)\right] \qquad (2.1)$$

onde p(x) é a probabilidade de um elemento x ocorrer e X é o conjunto de elementos.

Continuando com o exemplo, na Figura 2.1, temos o gráfico da entropia para uma variável aleatória binária, assumindo probabilidades de 0% até 100%.

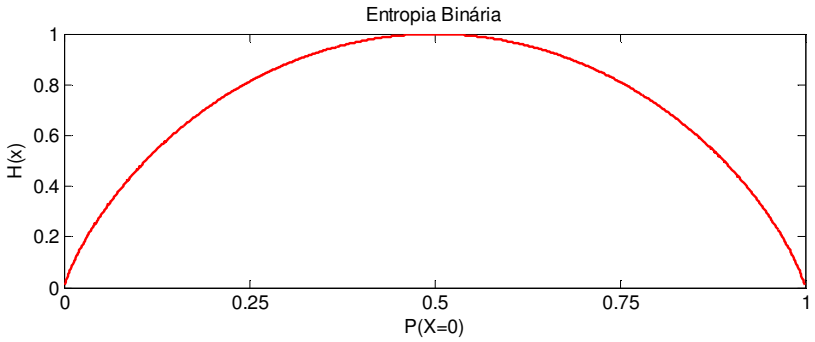

Figura 2.1 – Gráfico da Entropia de uma VA binária em relação à probabilidade de x ser igual a zero.

As questões mais importantes a serem observadas no gráfico são:

- Quando p(x=0) é igual a 0% ou 100%, a entropia é nula.

- O ponto máximo da função ocorre quando p(x=0) assume o valor 50%.

Com o auxílio do gráfico e de conhecimentos prévios, podem-se deduzir algumas propriedades, sem utilizar artifícios matemáticos. A primeira é a não negatividade, haja vista que não há informação negativa. Além disso, podemos analisar quando a entropia assumirá o valor máximo. Sabe-se que ela cresce com a incerteza da variável. Dessa forma, o ponto de maior incerteza é onde ocorrerá o ápice. Para uma VA binária, é fácil ver que isso ocorre quando as probabilidades de ser '0' ou '1' forem 50%. Ou seja, quando os elementos possíveis forem equiprováveis. É possível provar que o mesmo ocorre para qualquer que seja o número de variáveis. O valor máximo assumido pela entropia no caso geral é:

$$H_{máx}(X) = \log_2 |X| \tag{2.2}$$

onde $|X|$ é o número de elementos de X.

2.3 – Definições relacionadas à Entropia

A partir do conceito de entropia, faz-se a extensão para pares de VAs. Estas definições servem para o tratamento de dados correlacionados.

Como exemplo, podemos citar uma mensagem binária onde o primeiro bit (x) tem 50% de chance de ser '0'. Contudo, o bit em seguida (y) é, sempre, diferente do anterior. Nesse caso, o primeiro dígito carrega informação. Mas uma vez que se tem o primeiro bit, o seguinte não possui nada a acrescentar. Dessa forma, a **Entropia Conjunta** seria igual a H(X) e a **Entropia Condicional** seria zero. Definindo as duas, matematicamente, temos:

- **Entropia Conjunta:**

$$H(X;Y) = -E_p\left[\log_2 p(x;y)\right] = -\sum_{x\epsilon X}\sum_{y\in Y} p(x;y).\left[\log_2 p(x;y)\right] \qquad (2.3)$$

- **Entropia Condicional:**

$$H(X|Y) = -E_p\left[\log_2 p(x|y)\right] = -\sum_{x\epsilon X}\sum_{y\in Y} p(x;y).\left[\log_2 p(x|y)\right] \qquad (2.4)$$

Mais uma definição relacionada à entropia é a **Entropia Relativa [**$D(p||q)$**].** Ela é utilizada para medir a quantidade de bits a mais que devem ser usados, ao se assumir que a distribuição é q(x) quando, na verdade, ela é p(x). Ou seja, uma variável aleatória X de distribuição p(x) necessita de pelo menos $H(X)$ bits para ser descrita. Caso seja modelada com q(x), ela precisará de $H(X) + D(p||q)$. Para fins de compreensão, ela é considerada uma distância. Porém, isso não é verdade já que, normalmente, $D(p||q) \neq D(q||p)$ [5]. Entropia relativa é definida da seguinte forma:

$$D(p||q) = -E_p\left[\frac{\log_2 p(x)}{\log_2 q(x)}\right] = -\sum_{x\epsilon X} p(x).\left[\log_2\left[\frac{\log_2 p(x)}{\log_2 q(x)}\right]\right] \qquad (2.5)$$

Por fim, assim como foi visto que a entropia mostra a quantidade de informação contida nos dados, podemos também saber o quanto de informação um dado carrega sobre outro. Ou seja, ao saber o valor de uma VA, quanto se sabe da outra. Essa grandeza é conhecida como **Informação Mútua** e é definida da seguinte forma:

$$I(X;Y) = E_{p(x;y)}\left[\log_2 \frac{p(x;y)}{p(x).p(y)}\right] = \sum_{x\epsilon X}\sum_{y\in Y} p(x;y).\left[\log_2 \frac{p(x;y)}{p(x).p(Y)}\right] \quad (2.6)$$

$$I(X;Y) = E_{p(x;y)}\left[\log_2 \frac{p(y|x)}{p(y)}\right] = \sum_{x\epsilon X}\sum_{y\in Y} p(x;y).\left[\log_2 \frac{p(y|x)}{p(y)}\right] \quad (2.7)$$

Da mesma forma que a entropia, a informação mútua, entre duas VAs, não pode assumir valores negativos. A justificativa é igual: Não existe informação negativa. No pior dos casos, uma variável é independente da outra e a informação mútua será zero [5]. Além disso, seu valor não pode ultrapassar a entropia de X ou Y. Afinal, as duas VAs não podem conter, ao mesmo tempo, mais informação do que uma delas sozinha. Podem-se ver, também, as relações que a informação mútua tem com a entropia e as outras grandezas decorrentes [5]:

$$I(X;Y) = H(X) - H(X|Y) \quad (2.8)$$

$$I(X;Y) = H(X) + H(Y) - H(X;Y) \quad (2.9)$$

Pode ser visto que as propriedades das grandezas expostas, para duas VAs, são iguais às dos conjuntos cartesianos. Assim sendo, para simplificar a compreensão, utiliza-se o diagrama de Venn que representa as grandezas que foram anteriormente definidas e suas relações. Ele é mostrado na Figura 2.2.

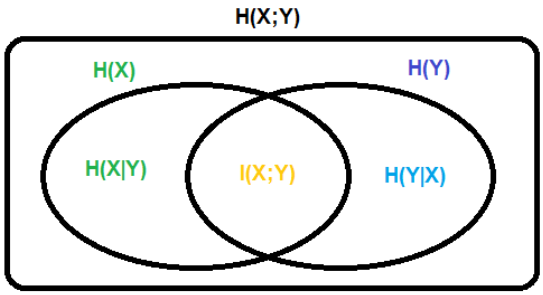

Figura 2.2 - Diagrama de Venn relacionando as entropias, entropias condicionais, entropia conjunta e informação mútua de duas VAs.

Assim como expandimos os conceitos para duas VAs, podemos fazê-lo para mais de duas. Para tal, utilizamos a regra da cadeia. A seguir, ela é exposta para três casos [5]:

- Entropia:

$$H(X_1; X_2; \dots; X_n) = \sum_{i=1}^{n} H(X_i | X_{i-1}; X_{i-2}; \dots; X_1) \qquad (2.10)$$

- Informação mútua:

$$I(X_1; X_2; \dots; X_n; Y) = \sum_{i=1}^{n} I(X_i; Y | X_{i-1}; X_{i-2}; \dots; X_1) \qquad (2.11)$$

- Entropia relativa:

$$D(p(x; y)||q(x; y)) = D(p(x)||q(x)) + D(p(y)||q(y)) \qquad (2.12)$$

2.4 – Sistema de comunicação

Após fazer o estudo sobre entropia e suas derivadas, será visto, agora, suas aplicações em transmissões de dados. Mas, primeiro, deve-se modelar um sistema de comunicação. Shannon fez isto em seu artigo "Communication in the Presence of Noise" [6]. Até hoje seu modelo engloba qualquer sistema de comunicação. Rádio, televisão ou pombo correio. Tendo-o como base, o esquema da Figura 2.3 foi feito:

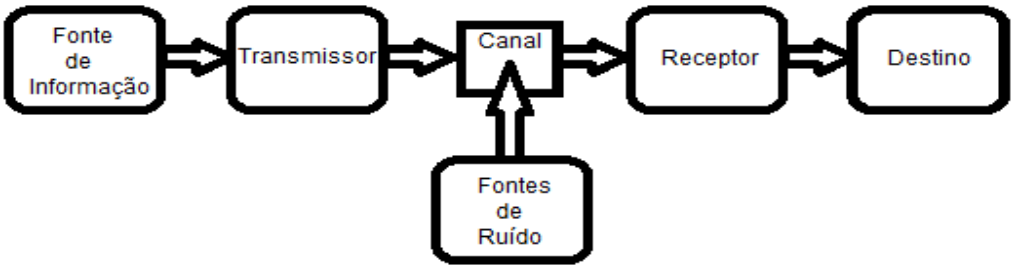

Figura 2.3 - Esquema de blocos geral de um sistema de comunicações.

Ele começa, naturalmente, com a **Fonte de Informação**. É deste ponto que surgem os dados a serem enviados. Essa fonte seleciona um grupo de símbolos dentre um alfabeto que compõem a mensagem. Exemplos: Pessoa, computador, emissora de rádio.

Toda a informação a ser enviada com todas as redundâncias necessárias para a sua compreensão compõe a **Mensagem**. Exemplos: Frase, imagem, música.

Em seguida, vem o **Transmissor**. Ele opera a mensagem de tal forma que o envio através do canal se torne possível e eficiente. Exemplos: Boca, modem, antena.

O meio por onde os dados são enviados é chamado de **Canal**. Exemplos: Ar, fio, vácuo.

Todo canal está sujeito a **Fontes de Ruído**. Podem ser de diversas naturezas. Devido a essas fontes, ocorrem distorções no sinal e, consequentemente, podem ocorrer erros na mensagem enviada. Exemplos: Barulhos, ondas eletromagnéticas, reflexões.

O **Receptor** capta a mensagem do canal e, se necessário, a decodifica. Exemplos: Ouvidos, modem, antena.

Por fim, a mensagem alcança o **Destino**. Exemplos: Pessoa, computador, rádio.

2.5 – Codificação de Canal

Essa parte da Teoria da Informação trata do problema de canais com ruído. Como foi visto, um canal ruidoso pode causar a alteração, ou mesmo a perda, de parte da mensagem. Em uma rede sem fio, por exemplo, essa perda chega a 10% da mensagem. Na maioria das ocasiões, tal nível de confiabilidade é inadmissível [7].

Na maioria dos casos, é impossível, ou muito custoso, retirar as fontes de ruído. Logo, o grande desafio está na eliminação dos seus efeitos, fazendo com que o receptor capte a mensagem enviada por um canal ruidoso. A fim de se garantir, então, que não ocorram perdas, utiliza-se a codificação de canal.

O conceito por trás da codificação é: Para poder verificar se o canal provocou erros e, em alguns casos, tentar corrigir esses erros, são inseridas redundâncias na mensagem. Naturalmente, essa inserção ocorre de forma controlada e conhecida. Assim, o receptor é capaz de decodificar a mensagem.

Um dos exemplos mais conhecidos de redundância inserida é o bit de paridade. Ele diz se o número de elementos iguais a ‘1’, na mensagem, é par ou ímpar. Contudo, é um método fraco. Primeiro, porque não proporciona correções. Segundo, porque protege a mensagem apenas caso haja um número ímpar de bits trocados. Caso o número seja par, o bit de paridade indicaria que não houve erros na mensagem, o que não seria verdade.

Outro exemplo de codificação que não corrige erros, mas os detecta, é o “Dois de cinco” (*Two-out-of-five*). Em um grupo de cinco bits utilizam-se sempre dois '1' e três '0'. Assim sendo, 10 combinações tornam-se possíveis. Por essa razão, ele é utilizado para representar dígitos decimais. Seu funcionamento é simples. Cada bit tem um peso. Eles são, respectivamente: 0 – 1 – 2 – 3 – 6. Para obter o valor do dígito representado, somam-se os pesos dos bits iguais a um. Por exemplo, 01001 representa 1 + 6 = 7. Apenas a representação de zero não atende isso: 01100. Dessa forma, a única maneira de se representar o número três torna-se 10010. Esse tipo de código é utilizado, por exemplo, em códigos de barras. Podemos ver que ele detecta qualquer erro de um bit, pois é necessário que sejam recebidos dois '1' e três '0'. Além disso, ele também detecta erros que ocorram em dois bits iguais. Isso o torna

mais robusto do que o bit de paridade, por exemplo. Porém, ele também não faz correção de erros.

Outra codificação que é clássica, na literatura, é a por repetição. No caso de ordem três, ao invés de se mandar apenas um bit, são enviados três bits iguais. Dessa forma, mesmo que um deles seja trocado no canal, o que foi enviado será conhecido. Ele não é a prova de erros, no entanto. Caso ocorra a mudança de dois, de um grupo de três bits, será recebida uma mensagem errada. Há os códigos de ordem mais alta, logo, mais robustos. Nestes, se repete o bit 5, 7, 9, (...) vezes. O aumento de dimensão, contudo, acarreta em uma taxa menor de informação enviada. Como é desejável utilizar a maior taxa possível, não se pode elevar em demasia o número de repetições. Assim sendo, vê-se que a taxa também é um fator importante quando se avalia um código. Ela equivale à razão entre o número de bits na mensagem original e o da mensagem codificada:

$$r = \frac{|\text{Mensagem Original}|}{|\text{Mensagem Codificada}|} \tag{2.13}$$

No exemplo do código de repetição, a taxa será sempre 1/n. Onde n é a ordem do código. Na Figura 2.4 tem-se um exemplo da codificação de ordem três. Nela, uma entrada de seis bits gera uma saída de dezoito.

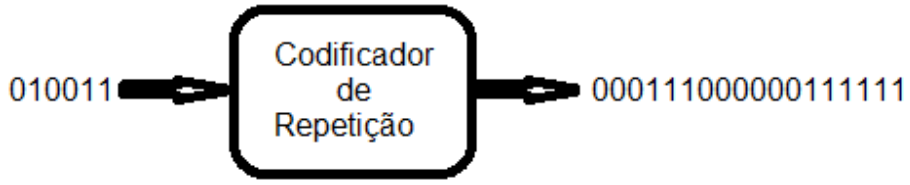

Figura 2.4 - Exemplo de código com repetição de ordem três.

Um exemplo de codificação mais inteligente é o Código de Hamming. Ele possui uma taxa melhor do que a do código de repetição. Além disso, corrige erros de um bit e é capaz de detectar erros de dois. Para compreendê-lo, é preciso, primeiro, o conceito de distância de Hamming: No caso de uma palavra binária, esta distância é o número de bits que mudam entre as diferentes mensagens. Por exemplo, a distância entre as mensagens em uma codificação por bit paridade é dois. Afinal,

caso dois bits mudem, a mensagem torna-se uma diferente, mas válida (sem erros detectáveis). Já no caso de um código de repetição de ordem três, a distância é três, uma vez que a mensagem só seria válida caso mudassem os três bits. Contudo, como ocorre uma correção por aproximação, começam a ocorrer erros a partir do momento em que o erro ultrapassa a metade da distância. Ou seja, quando ocorrem dois erros.

Sabendo disso, Hamming quis utilizar o conceito no bit de paridade, mas aumentando a distância entre as mensagens. Ele queria, também, aumentar a taxa do código. A idéia, então, foi usar grupos de bits de paridade. Primeiro, faz-se que erros em elementos diferentes provoquem mudanças diferentes nos bits de controle de erro. Então, com três bits é possível averiguar erros em uma mensagem de dimensão igual a sete. Para tal, Hamming interpolou os bits de paridade. Assim sendo, cada dígito da mensagem é englobado por um diferente grupo deles. A idéia fica mais clara no esquema da Figura 2.5:

Posição		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
Bit		p1	p2	d1	p3	d2	d3	d4	p4	d5	d6	d7	d8	d9	d10	d11	p5	d12	d13	d14	d15
Cobertura do bit de paridade	p1	X		X		X		X		X		X		X		X		X		X	
	p2		X	X			X	X			X	X			X	X		X	X		
	p3				X	X	X	X					X	X	X	X					X
	p4								X	X	X	X	X	X	X	X					
	p5																X	X	X	X	X

Figura 2.5 - Cobertura dos bits de paridade em um código de Hamming .

Apesar de poder ser estendida para mensagens de diversos tamanhos, o mais utilizado é o modelo com sete bits onde três são de paridade. Nesse caso, o gráfico da Figura 2.6 mostra quais bits cada bit de correção de erro engloba:

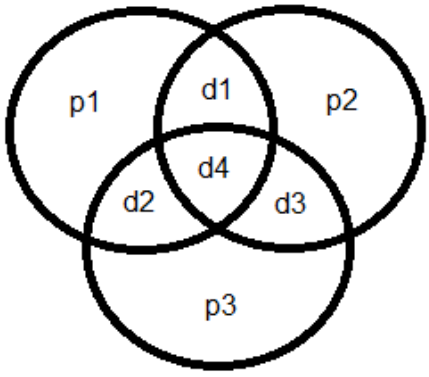

Figura 2.6 – Cobertura dos bits de paridade de um código de Hamming com sete bits.

Para facilitar a geração do código, utiliza-se uma matriz de geração G_{4x7}:

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

Monta-se, primeiramente, uma matriz M_{1x4} com a mensagem. Então, faz-se a operação XOR entre M e G. O resultado é uma matriz 1x7 com a mensagem a ser enviada. Ela é composta da mensagem original concatenada com os bits de paridade.

Por fim, para se realizar a verificação e descobrir se houve erro e onde ele ocorreu, utiliza-se a matriz H_{3x7} de verificação de paridade:

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

Basta, então, criar uma matriz R_{7x1} com a mensagem recebida. Então, faz-se outro XOR entre H e R. O resultado será uma matriz 3x1. Ela corresponderá a um número binário indicando a posição do bit que foi trocado.

O código de Hamming no caso (7.4) possui uma taxa de 4/7. Ou seja, uma mensagem de quatro dígitos binários, após ser codificada, torna-se uma mensagem de sete bits. Além disso, possui uma distância de Hamming de três bits. Afinal, uma mensagem recebida errada somente será considerada válida caso três dos sete bits tenham sido trocados. Portanto, realmente ocorreram as melhoras que Hamming tinha como objetivo.

A codificação de Hamming não é, no entanto, o melhor que se pode alcançar. Afinal, esse código foi desenvolvido na década de quarenta e ainda hoje são feitos estudos sobre codificação de canal. Então, a fim de se fazer códigos mais robustos e eficientes, um maior estudo da teoria por trás do envio de informação através de um canal é necessário.

2.6 – Capacidade de Canal

Tendo em vista que a informação pode ser medida pela entropia, resta, apenas, descobrir a taxa de informação que se consegue mandar em um determinado canal. Shannon deduz matematicamente esse limite e o denomina **Capacidade de Canal**.

O desafio tornou-se, portanto, descobrir a codificação que faz com que a taxa seja a mais próxima possível do limite calculado. Seguem-se, então, os enunciados do **Teorema de Shannon**:

- Um dado sistema de comunicação tem uma taxa máxima de informação denominada **Capacidade de Canal** (C).

- Se a taxa de informação enviada for menor do que C, então, é possível obter um erro tão pequeno quanto se queira, utilizando técnicas de codificação inteligente

Estes resultados de Shannon dizem respeito a um canal sem memória. Isto é, um canal onde a probabilidade da entrada atual independe das entradas anteriores e, da mesma forma, a probabilidade da saída atual independe das saídas anteriores. Isso é verdade para a maioria dos canais. Assim sendo, os resultados atingidos por ele podem ser aplicados sem muita preocupação quanto ao sistema de comunicação que está sendo utilizado.

Shannon calcula, por fim, a capacidade de canal. O limite teórico para o envio de informação através de um dado canal. Tem-se um sistema com um alfabeto de entrada X, um de saída Y e a matriz de probabilidade de transmissão p(y|x). A **Capacidade de Canal** é definida como a maximização da informação mútua entre X e Y, variando-se p(x). Matematicamente, temos:

$$C = \max_{p(x)} I(X;Y) \tag{2.14}$$

Como a capacidade de canal vem da informação mútua, entre duas VAs, sua primeira propriedade é não poder assumir valores negativos. Realmente, não faria sentido pensar em um canal no qual só seria possível passar informação negativa, pois essa não existe. Da mesma forma, seu valor não pode ultrapassar a entropia máxima de X ou de Y. Esse é o maior valor que pode assumir a informação mútua dos dois. Caso fosse possível atingir valores maiores, seria como se o canal inserisse informação sobre X ou Y na mensagem. Naturalmente, isso não ocorre.

É interessante, também, pensar em canais com realimentação. Seria normal pensar que um retorno após o canal possibilitaria uma comunicação mais eficiente. Entretanto, é possível provar que não [5]. Tendo-se um canal de capacidade C, supõe-se que seja instalada uma realimentação. Com ela, o transmissor sabe qual foi a mensagem recebida. Ainda assim, a capacidade do canal permanece inalterada. Ou seja, o seu limite de envio de informação não aumentaria, como poderia ser esperado.

Para exemplificar o cálculo de capacidade de canal, será utilizado um canal binário simétrico. Este possui certa probabilidade p de trocar o valor do bit durante o envio de mensagem. Ou seja, caso seja enviado '0', a chance deste bit não ser alterado para '1' é 1-p enquanto a chance de ele se tornar '1' é p. Da mesma forma o inverso. A chance de ser enviado '1' e de ser recebido '1' é 1-p enquanto há uma probabilidade p de ser recebido '0' pelo receptor. O esquema da Figura 2.7 ilustra esse canal:

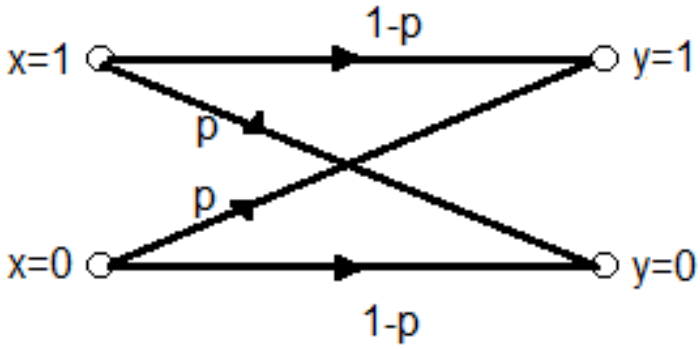

Figura 2.7 - Esquema de canal binário simétrico.

O cálculo da capacidade desse tipo de canal é feito em "Sistemas de Comunicação Analógicos e Digitais" [7]. A fim de descobri-la, precisamos da informação mútua máxima. Para tal, a entropia também tem de ser máxima. Como

foi visto na seção 2.2, isso ocorre com a distribuição equiprovável. Para uma VA binária, temos p(x=0) = p(x=1) = 50%. Por simetria, podemos afirmar, também, que p(y=0) = p(y=1) = 50%. Dessa forma, utiliza-se, por fim, a equação (2.6), com a fórmula do valor esperado para calcular a máxima informação mútua:

$$\sum_{k=0}^{1}\sum_{j=0}^{1} p(x = j\ ;\ y = k).\log_2\left[\frac{p(y = k \mid x = j)}{p(y = k)}\right]$$

Substituindo-se os valores pelas probabilidades iguais a 50% e as probabilidades condicionadas por p e 1-p, segundo o caso, temos:

$$C = 1 + p.\log_2(p) + (1 - p).\log_2(1 - p) \qquad (2.15)$$

Utilizando a definição de entropia feita em (2.1), temos:

$$C = 1 - H(p) \qquad (2.16)$$

Dois casos são interessantes de serem analisados:

- Se p = 0, $H(p)$ torna-se zero e C, um. Este é seu valor máximo. Nesse caso, é possível o envio de um bit para cada bit de informação, medida pela entropia. Este seria o caso ideal. É interessante notar que no caso em que p = 1, C também é um. Isto faz sentido, pois, se é certa a transformação do bit, basta transformar toda a mensagem recebida para se ter o conteúdo enviado.

- Se p = 50%, $H(p)$ torna-se um e C, zero. Este é seu valor mínimo. Nesse caso, um bit enviado não contém nenhum bit de informação. Esse canal é inutilizável para o envio de informação. Isto decorre da completa independência que a saída possui com a entrada.

Sabe-se, agora, o limite principal da Teoria da Informação. Este é, talvez, o resultado mais importante do trabalho de Shannon. A partir do conhecimento dos limites, os engenheiros que trabalham com telecomunicações tentam, cada vez mais, alcançá-lo. Isto porque o trabalho de Shannon não é completo. Primeiro, ele demonstra que existe um limite para a taxa de informação. Então, o calcula. Depois, mostra que para alcançar este limite, basta utilizar uma codificação inteligente. Por fim, ele demonstra que cada caso possui, necessariamente, ao menos um código ideal. Contudo, suas fórmulas não indicam qual é esse código. Com isso, abre-se um campo onde se sabe qual o melhor resultado e que é possível atingi-lo. Não se sabe, entretanto, qual o meio para fazê-lo.

2.7 – Canal com apagamento

Nesse projeto, o trabalho será desenvolvido para canais com apagamento. Esse tipo de canal foi introduzido por Elias em "Coding for two noisy channels" [8]. Estes canais são caracterizados pela perda de um percentual de dados. No caso de um canal binário, os dados perdidos serão bits. Não ocorre, entretanto, mudança dos dados. Esse tipo de canal tem, portanto, duas entradas ('0' ou '1') e três saídas ('0', '1' ou erro). O esquema da Figura 2.8 ilustra esse tipo de canal:

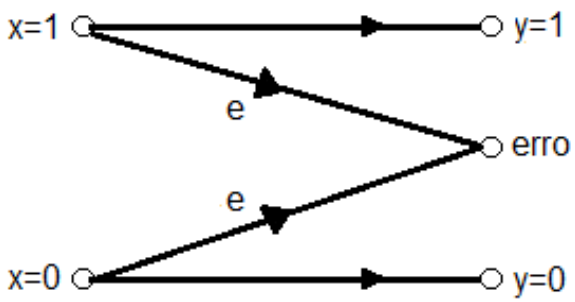

Figura 2.8 - Esquema de canal binário com apagamento.

Esse tipo de modelagem pode ser usado para canais ruidosos onde os códigos detectores de erro sejam suficientemente robustos. É o caso, por exemplo, de redes de internet. Nesse exemplo, os dados encapsulados estão bem protegidos e caso ocorra erro, não há troca de bit.

O cálculo de capacidade de canais com apagamento é feito em "Elements of Information Theory" [5]. Para tal, recorre-se à propriedade da informação mútua da fórmula (2.7). Logo,

$$\mathrm{C} = \max_{\mathrm{p(x)}}[\mathrm{H(Y)} - \mathrm{H(Y|X)}] = max_{p(x)}[H(Y) - H(e)] \qquad (2.17)$$

Tem-se, também, que:

$$\mathrm{H(Y)} = \mathrm{H}[(1-\mathrm{p})(1-\mathrm{e}); \mathrm{e}; \mathrm{p}.(1-\mathrm{e})] = \mathrm{H(e)} + (1-\mathrm{e}).\mathrm{H(p)} \qquad (2.18)$$

Assim sendo, juntando-se (2.16) e (2.17), chega-se a:

$$\mathrm{C} = \max_{\mathrm{p(x)}}[(1-\mathrm{e}).\mathrm{H(p)}] \qquad (2.19)$$

E, por fim, como no caso binário a entropia máxima é igual a um:

$$\mathrm{C} = (1-\mathrm{e}) \qquad (2.20)$$

Esse resultado é intuitivo. Como ocorre erro em um percentual *e* da mensagem, é natural imaginar que, no melhor dos casos, se aproveite tudo exceto o que foi perdido. Como este trabalho visa estudar canais com apagamento, esse valor calculado torna-se uma referência importante a ser utilizada.

Foi visto, portanto, como medir a informação utilizando a entropia e falou-se, também, sobre informação mútua. Além disso, para analisar sistemas de comunicação mais profundamente, foram explorados alguns códigos de canal e, em seguida, a capacidade de canal. Lembra-se que o objetivo de qualquer codificação é atingir uma taxa a mais próxima possível desse valor. Por fim, foi exposto e analisado o canal com apagamento, que será o utilizado nesse trabalho. Todas as codificações que foram vistas possuíam uma taxa fixa. Entretanto, existem aqueles cuja taxa pode variar. No capítulo seguinte fala-se sobre eles. São expostas suas vantagens, utilidades e, por fim, sua implementação.

Capítulo 3

Códigos Fontanais

3.1 – Fontanas Digitais

O conceito de Fontana Digital foi introduzido em "A Digital Fountain Approach to Asynchronous Reliable Multicast" [9] e é bastante recente (2002). A analogia é uma fonte de água que fornece quantas gotas forem necessárias, até que se encha um copo. Se algumas dessas gotas caírem fora do recipiente, ainda assim será possível enchê-lo com as gotas seguintes.

Trazendo para a área de telecomunicações, o transmissor possui uma quantidade suficientemente grande de pacotes codificados (fonte com água). Ele os envia, até que toda a mensagem tenha sido recebida (o copo tenha ficado cheio). Caso alguns dos dados enviados sejam perdidos (gotas fora do recipiente), ainda assim será possível receber a mensagem com os próximos pacotes. Uma definição formal de Fontana Digital Ideal é encontrada em "Códigos Fontanais Bidimensionais para Canais com Apagamento" [10] que cita Mitzenmacher [11]:

Uma Fontana Digital Ideal é um esquema capaz de produzir, a partir de k símbolos da mensagem de entrada s, uma sequência de símbolos codificados t que obedece às propriedades,

- $|t|$ é um número arbitrariamente grande (possivelmente infinito) de símbolos codificados. (Idealmente, dada a mensagem original s, cada símbolo codificado t_i pode ser formado em um tempo constante).

- Existe um receptor capaz de reconstruir a mensagem original s de tamanho k, a partir de qualquer conjunto $\{t_1; t_2;...; t_k\}$ de k símbolos codificados. (Essa reconstrução deve ser rápida, de preferência linear em k).

O principal conceito por trás das Fontanas Digitais é, portanto, a possibilidade de reconstruir a mensagem com qualquer grupo de pacotes, desde que ele tenha tamanho suficiente. Há diversas áreas onde isso é interessante [11]:

- *Multicast* – O conceito de Fontana Digital foi criado para tornar *multicasts* mais confiáveis. Nesse caso, os pacotes são enviados para todos os usuários conectados à fonte de informação, até que o usuário se desconecte. A perda de dado que ocorra com um destinatário não é refletida, então, nos outros.

- Recepção paralela de arquivo – Da mesma forma que o *multicast,* pode enviar para diversos lugares, pode-se receber uma mensagem de diversos lugares, acelerando o processo.

- *One-to-Many* TCP – Também tem um raciocínio semelhante ao *multicast.* Nesse caso, contudo, é orientado para a arquitetura TCP

- Vídeo em tempo real – Ao segmentar o arquivo de vídeo em diversas partes, o tempo de espera para recebimento dos dados não aumenta. Enquanto o usuário assiste um bloco, os seguintes são carregados.

- Transmissão em Redes Sobrepostas – Ao associar diversas redes, torna-se mais complicado o controle de fluxo. Assim sendo, torna-se mais fácil o uso de Fontanas Digitais para troca de dados.

Um aspecto interessante a se notar nesse tipo de código é o fato de ele não possuir uma taxa definida. É denominado código evanescente (*rateless code*). Uma vez que não se sabe quantos pacotes serão apagados, não é possível saber a quantidade de bits que serão enviados para cada mensagem. Devido a esse fato, seu desempenho deve ser calculado sobre a média de um grupo de eventos, e não em apenas um caso.

3.2 – Códigos LT

Códigos LT foram a primeira realização viável de um código de taxa variável. Eles foram propostos por Michael Luby em 2002 no artigo "LT codes" [12]. Eles se baseiam em operações XOR e, dessa forma, possuem codificadores e decodificadores rápidos.

3.2.1 – Codificação

O codificador proposto por Luby recebe, naturalmente, a mensagem a ser enviada **M**. Ela é composta de um conjunto de bits $\{m_1; m_2;...; m_k\}$, onde k é o tamanho da mensagem. Sua saída é **X**, composta pelo conjunto de bits $\{x_1; x_2;...\}$ onde sua dimensão é tão grande quanto for necessário. Cada um dos elementos de **X** é uma combinação de um subconjunto de **M**. Essa combinação é feita com operações XOR. A escolha dos subconjuntos ocorre da seguinte forma [12]:

- Segundo uma distribuição pré-determinada, denominada **Distribuição de Graus LT**, define-se o grau (d_n) de cada bit x_n. A escolha apropriada da distribuição será discutida mais a frente.

- Segundo uma distribuição uniforme, escolhe-se, aleatoriamente, d_n bits de **M**. A operação XOR entre todos os elementos desse subconjunto gerará x_n.

Esses procedimentos irão gerar um grafo. Ele irá relacionar a entrada e a saída do codificador. Cada elemento de **X** assumirá o valor da operação XOR feita em todos os elementos de **M** que apontem para ele.

Para melhor compreensão, segue-se um exemplo [10]:

M = {1; 0; 0; 1; 1}

Saída	Grau	Bits utilizados	Valor
x1	2	m1; m2	m1⊕m2 = 1
x2	3	m1; m2; m4	m1⊕m2⊕m4 = 0
x3	4	m1; m2; m3; m5	m1⊕m2⊕m3⊕m5 = 0
x4	4	m1; m3; m4; m5	m1⊕m3⊕m4⊕m5 = 1
x5	1	m5	m5 = 1
x6	2	m2; m5	m2⊕m5 = 1

Tabela 3.1 - Exemplo de codificação LT.

X = {1; 0; 0; 1; 1; 1}

O grafo gerado é visto na Figura 3.1:

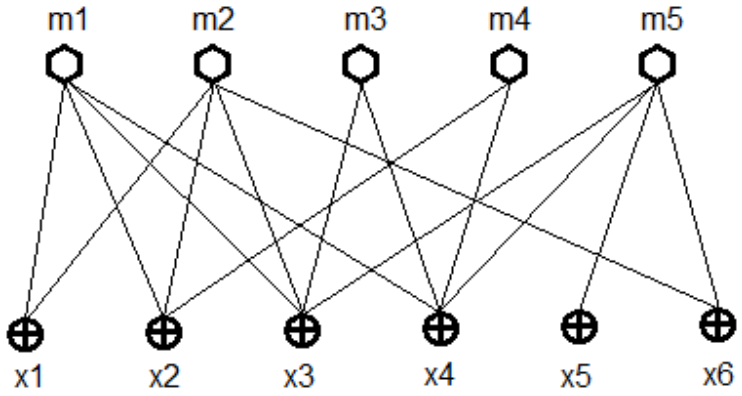

Figura 3.1 - Grafo de exemplo de codificação LT.

Podemos notar que cada elemento de **X** é formado fazendo-se a operação XOR entre os bits ligados a ele. A partir do grafo, pode-se, finalmente, criar a matriz **G** que relaciona a entrada e a saída do decodificador. Nela, as linhas representam as saídas e as colunas, as entradas.

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

3.2.2 – Decodificação

Para decodificar a mensagem recebida, supõe-se que a matriz **G** é conhecida pelo receptor. Assim sendo, o problema de decodificação torna-se similar a uma solução de sistema de equações. Procura-se um x_i que tenha grau um. Dessa forma, ele possui valor igual a m_j, determinado pela matriz **G**. Com um bit da mensagem original conhecido, é possível diminuir um grau dos demais elementos de **X** que o continham. Tal procedimento é repetido até que não haja mais bits a serem desvendados.

Dando continuação ao exemplo [10], sua decodificação seria feita da seguinte forma, mostrada na Figura 3.2:

1. Na primeira iteração, o único nó de verificação conectado a apenas um símbolo de entrada é x_5.
2. Faz-se então $m_5 = x_5$ e elimina-se o nó de verificação x_5.
3. Adiciona-se o valor de m_5 aos nós de verificação que estão ligados ao mesmo, desconectando m_5 do grafo.
4. No inicio da segunda iteração, o sexto nó de verificação está ligado apenas a m_2. Faz-se então $m_2 = x_6$ e elimina-se o nó de verificação x_6.
5. Adiciona-se o valor de m_2 aos três nós de verificação ligados a ele, desconectando m_2 do grafo.
6. No inicio da terceira iteração, o primeiro nó de verificação está ligado apenas a m_1. Faz-se então $m_1 = x_1$ e elimina-se o nó de verificação x_1.
7. Adiciona-se o de valor m_1 aos três nós de verificação ligados a ele, desconectando m_1 do grafo.
8. No inicio da quarta iteração, escolhe-se o segundo nó de verificação que está ligado apenas a m_4. Faz-se então $m_4 = x_2$ e elimina-se o nó de verificação x_2.
9. Adiciona-se o valor de m_4 ao nó de verificação restante ligado a ele, desconectando m_4 do grafo.
10. Finalmente, percebe-se que dois nós de verificação estão ligados a m_3, e ambos atribuem o mesmo valor a tal símbolo de entrada, o qual é restaurado.

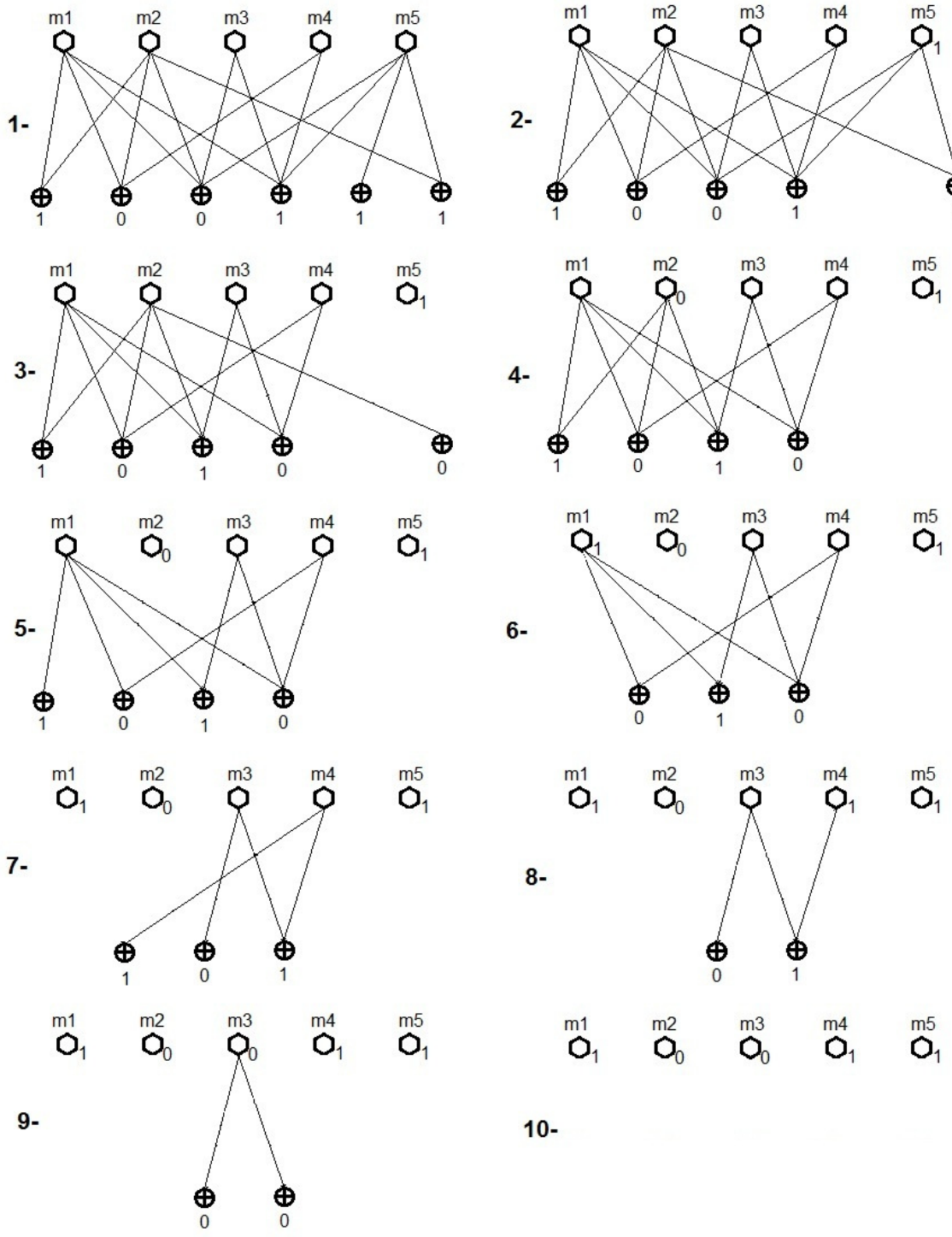

Figura 3.2 - Exemplo de decodificação de código LT.

É possível notar que, inicialmente, nenhum dos elementos da mensagem original está **coberto**. Isto é, não há nenhum x_i de grau unitário ligado a ele. Então, são recebidos bits codificados até que surja algum, cujo grau seja um. O símbolo m_j equivalente é então processado. Ou seja, ele assume o valor do bit recebido e passa por uma operação XOR com todos os elementos de **X** ligados a ele. Dessa forma, é possível que ocorra o surgimento de novos bits que possam ser processados. Um grupo de bits que está coberto mas ainda não sofreu processamento é denominado ***ripple***.

Com esse exemplo, é possível perceber como a decodificação pode ocorrer rapidamente. Esta velocidade dependerá, principalmente, do tamanho do bloco de dados a ser decodificado. Ele definirá o número de iterações a serem feitas. Já que as operações de XOR são pouco custosas, computacionalmente, o número de repetições será o fator determinante para definir a velocidade.

Poderia, no entanto, ser levantado um problema. A mensagem **X** enviada depende de uma distribuição probabilística. Assim sendo, como o decodificador pode saber quais os graus dos elementos de **X**? Como ele pode saber, também, quais os subconjuntos de **M** que se relacionam com cada elemento de **X**? Enfim, como o receptor tem acesso à matriz **G**?

Luby sugere uma solução [14]. Essa informação pode ser explicitamente contida no pacote de dados enviado. Ou seja, a coluna da matriz **G** poderia ser incluída no cabeçalho, por exemplo, como um metadado. Outra solução seria replicar o processo pseudo-aleatório que gerou a matriz utilizada. Para isso, bastaria enviar, no pacote, a semente utilizada no processo e a posição do elemento em **X**. Tomando-se alguma dessas precauções, o receptor torna-se capaz de produzir a matriz **G**. Com ela, a decodificação pode ser feita.

É interessante notar que os códigos LT escapam da definição formal de Fontana Digital. Neles, é necessário um número de bits maior do que a mensagem original para que se faça a decodificação. Contudo, para uma mensagem longa, esses números tornam-se muito próximos. Considera-se, então, que os códigos LT são uma boa aproximação de Fontana Digital.

3.3 – Implementação

A implementação do código de Luby foi feita através do programa Matlab. Os algoritmos utilizados para codificação e decodificação têm como base os propostos em "Optimizing the Degree Distribution of LT Codes with an Importance Sampling Approach" [15]. Entretanto, para facilitar a análise futura, alguns detalhes foram modificados. Eles estão expostos a seguir:

- Algoritmo de Codificação:

  Repetir,

  1 – Determinar o grau da saída x_i.

  2 – Determinar o subconjunto de **M** a ser utilizado em x_i.

  3 – Construir **X**.

  4 – Construir matriz **G.**

  Até i = n.

  5 – Enviar **X** e **G.**

onde n é vinte vezes o tamanho da mensagem. Esse número foi escolhido por ser suficientemente grande. Dessa forma, evitam-se erros de leitura, mesmo com uma distribuição de graus ruim e um canal com muito apagamento.

- Algoritmo de Decodificação

  Repetir

  Enquanto não houver bit com grau um, **faça**

  1 – Receber bit

  Fim enquanto

  2 – $m_i \leftarrow x_j$ (bit com grau um)

  Para todo G (i; k) = 1 **faça**

  3 - **G (i; k)** $\leftarrow$ **G (i; k)** $\oplus$ m_i

  Fim para todo.

  Até mensagem decodificada.

Além dessas duas funções, foi necessário, para os testes, emular um canal com apagamento. Isto foi feito com uma função que tem um percentual p de falhas, como entrada. Assim, ao passar pela função, cada bit da mensagem tem uma chance p de ser substituído por '2'. Caso contrário, ele permanece inalterado.

Por fim, os testes foram feitos utilizando uma função que chama as três anteriores. Esta funciona da seguinte forma: Primeiramente cria-se uma mensagem aleatória. Então, ela passa pelo codificador. Este retorna a mensagem codificada e a matriz **G**. Após isso, os dados codificados servem de entrada para o emulador de canal com apagamento. Ele retorna uma mensagem com erros. Finalmente, esta é enviada, junto com a matriz **G,** para o decodificador. Como retorno, ele tem o número de elementos de **X** utilizados para a decodificação. Este é o equivalente ao número de bits que teriam sido enviados em uma transmissão real. A taxa do código é, portanto, o tamanho da mensagem dividido por essa quantidade de bits.

Todo esse procedimento é repetido cem vezes, a fim de que seja encontrada uma taxa média, uma vez que ela é variável. Além disso, é possível controlar duas variáveis que influenciam os resultados. São elas: o tamanho da mensagem e a taxa de apagamento do canal.

São feitas simulações para diversos valores dessas duas variáveis. A partir de seus resultados, são feitos gráficos de desempenho para cada tipo de distribuição de graus. Através deles, então, as distribuições podem ser comparadas.

3.4 – Distribuições de Graus LT

A distribuição de graus é uma distribuição probabilística que varia de um até k. Sendo k a dimensão da mensagem. Ela determina a probabilidade de ocorrência de cada grau em uma codificação. A definição formal de distribuição de grau ($\rho(d)$) é feita por Luby em "LT codes" [12]:

- Para todo grau d, $\rho(d)$ é a probabilidade de um símbolo x_i tenha grau d.

É intuitivo notar que essa distribuição tem grande influência nos resultados de desempenho da codificação. Ela deve ser analisada com cuidado, pois é interessante que haja pacotes de graus baixos. Eles facilitam a decodificação. No entanto, graus altos também são necessários, uma vez que é preciso que todos os elementos da mensagem sejam abrangidos. Além disso, muitos bits de grau baixo trariam muita redundância.

Segundo Luby [12], há dois objetivos a serem atingidos por uma distribuição de graus:

- Possuir a menor quantidade possível de símbolos codificados (em média). Sendo essa quantidade o menor número de bits necessários para uma decodificação bem sucedida.

- Possuir símbolos codificados com o menor grau possível (em média). Ele determina o número de operações necessárias para se recuperar uma mensagem. Esse número é igual à multiplicação do grau médio com o número de bits recebidos.

Outra forma de se analisar o impacto da distribuição de graus, é através do *ripple*. Idealmente, a partir do momento em que são recebidos bits suficientes, ele não pode ser zerado. Se assim for, significa que será necessário o envio de novos dados. Contudo, deve ser evitado, também, que o *ripple* assuma dimensões muito elevadas. Caso isso ocorra, provavelmente existe muita redundância na mensagem. Isso significa, então, que a taxa será desnecessariamente alta.

Diversos padrões já foram testados para avaliação de desempenho. Nos subitens seguintes, alguns desses testes serão repetidos a fim de poderem ser usados como comparação, posteriormente.

3.4.1 – Distribuição de Grau Unitário

Essa é a distribuição mais simples possível. Nela, todos os bits têm grau unitário. Ou seja, $\rho(1) = 100\%$.

Assim sendo, cada x_i recebido é, imediatamente, convertido em um elemento de **M**. Essa transmissão é ainda pior do que uma sem codificação. Isto porque os símbolos enviados são escolhidos aleatoriamente. Dessa forma, não há como garantir que todos os símbolos sejam cobertos. Especialmente para mensagens muito longas, a probabilidade de haver bits que não são contemplados por nenhum elemento de **X** é muito alta.

Há ainda outro aspecto negativo. Mesmo para mensagens curtas, é preciso receber diversos bits desnecessários até que cheguem os certos. Com isso, é recebida muita informação redundante. A taxa, então, torna-se desnecessariamente alta. Em outras palavras, essa distribuição não respeita a regra de manter o *ripple* pequeno.

No gráfico da Figura 3.3 vêem-se todos esses efeitos. Mensagens curtas têm taxas muito altas. Por exemplo, para o envio de uma mensagem de dez bits, são recebidos, em média, 28,89 bits. A taxa é, portanto, 2,889. Essa é a menor de todas as analisadas. Este valor já está muito acima do que se espera para um código Fontanal. Além disso, pequenos aumentos no tamanho da mensagem implicam em grandes aumentos na taxa de bits enviados.

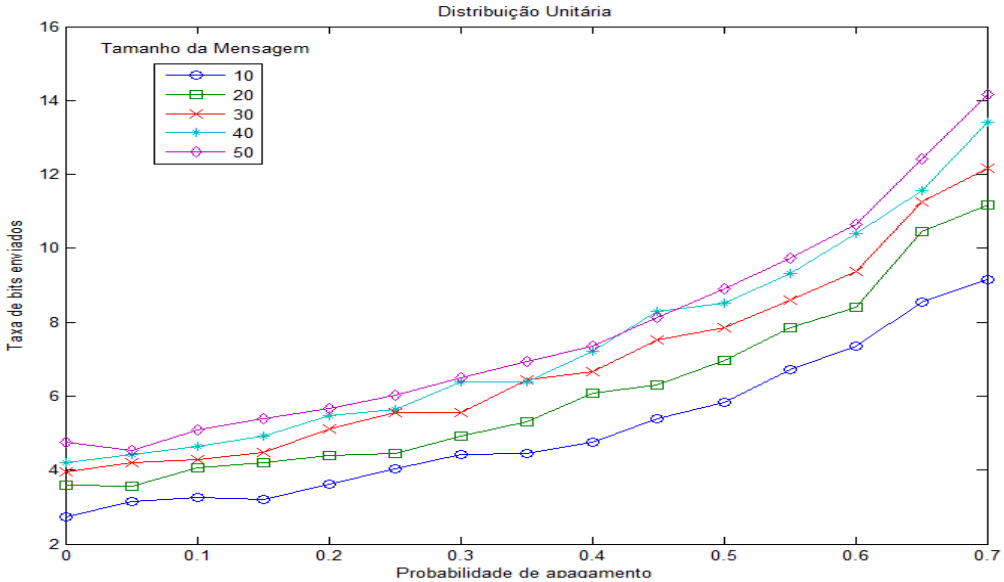

Figura 3.3 - Gráfico de taxa de bits enviados em relação à probabilidade de apagamento do canal para uma distribuição de grau unitário.

3.4.2 – Distribuição Equiprovável

O passo seguinte é uma distribuição onde todos os graus têm a mesma chance de ocorrer. Ou seja, $\rho(d) = 1/k$.

Nesse caso, o problema torna-se a dificuldade de decodificação. Isto porque, provavelmente, todos os símbolos serão abrangidos. Contudo, a probabilidade de haver elementos de **X** com graus baixos torna-se pequena. Assim sendo, principalmente à medida que aumenta o tamanho da mensagem, mais difícil fica a decodificação.

Isto é consequência de essa distribuição não respeitar a questão de não zerar o *ripple*. Dessa forma, há muita informação que não é acessada. Isso leva a uma necessidade ainda maior de bits para o descobrimento da mensagem original. Esse número cresce tanto que, diversas vezes, o receptor não é capaz de fazer a decodificação.

Por exemplo, na Figura 3.4, foi possível gerar resultados para mensagens de dimensão igual a, no máximo, quarenta bits. Para tamanhos superiores a esse, o percentual de mensagens não lidas torna-se muito alto.

O melhor desempenho também foi muito ruim. Para uma mensagem de dez bits em um canal sem apagamento, envia-se, em média, 31,64 bits. A taxa é de 3,164.

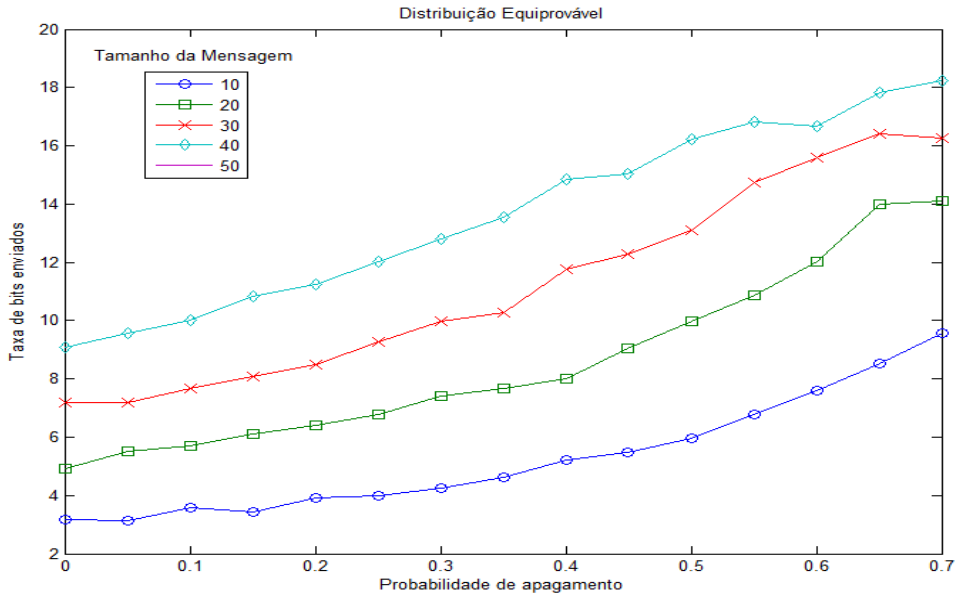

Figura 3.4 - Gráfico de taxa de bits enviados em relação à probabilidade de apagamento do canal para uma distribuição equiprovável.

3.4.3 – Sóliton Ideal

Para manter baixos o grau médio da mensagem codificada e o número de bits necessários para decodificá-la, Luby propõe o uso da distribuição **Sóliton Ideal** [12]. Ela pode ser visualizada na Figura 3.5 e é definida da seguinte forma:

- $\rho(1) = \frac{1}{k}$
- $\rho(i) = \frac{1}{i.(1-i)} \; \forall \, k: 1 < i < k$

onde k é a dimensão da mensagem.

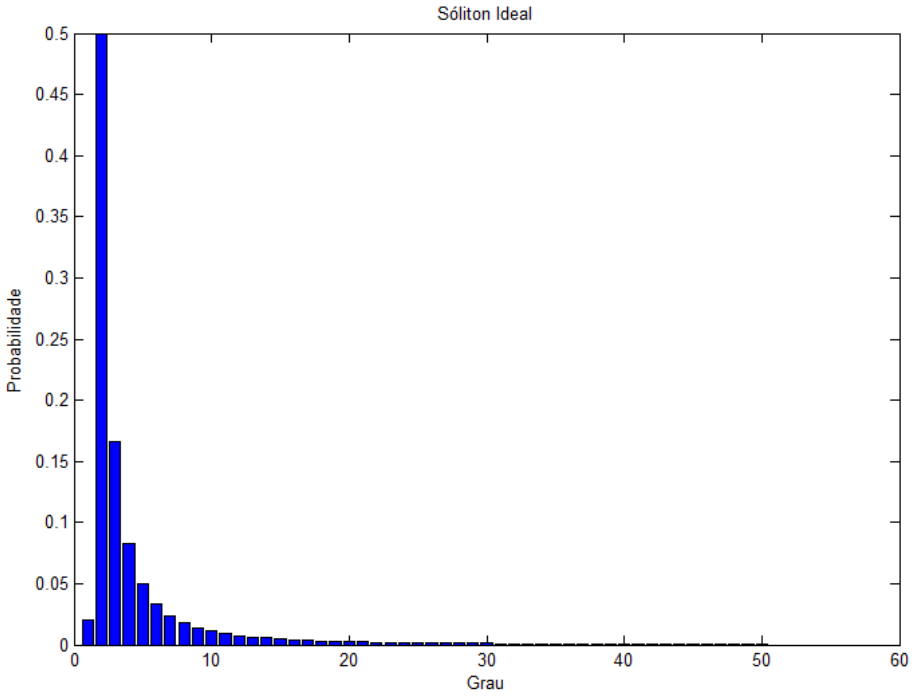

Figura 3.5 - Distribuição Sóliton Ideal.

Em termos de expectativa probabilística, esta distribuição deveria funcionar perfeitamente. Em seu comportamento ideal, ela faria com que o *ripple* fosse sempre igual a um, obedecendo às regras citadas por Luby. Assim, a decodificação poderia ser feita com exatamente k símbolos recebidos. Seria atingido, portanto, o limite da capacidade do canal.

Infelizmente, como quase tudo que é ideal, ela não tem bons resultados. Isso ocorre porque a análise heurística leva em consideração que o valor esperado de certa função será, realmente, o seu valor. Isso, contudo, não é verdade. Provavelmente, ocorrerão flutuações. Estas farão com que, em algum momento, o *ripple* seja zerado. Essas flutuações são ainda mais frequentes quando aumenta a taxa de apagamento do canal. Logo, ele consume mais bits codificados. Devido a isso, o processo de decodificação torna-se mais longo, também.

O resultado encontrado já é, no entanto, muito superior à distribuição de grau unitário e à equiprovável. Primeiramente, o aumento do tamanho da mensagem não acarreta grandes alterações no resultado. Além disso, dificilmente ocorrem erros na leitura da mensagem. E, ainda, as taxas atingidas são muito menores do que as anteriores. Para uma mensagem de dez bits em um canal sem apagamento, em média, enviam-se 31,64 bits ao se utilizar a distribuição uniforme. Para a de grau unitário, 28,89. Já para a Sóliton ideal, apenas 16,29. A taxa, portanto, é 1,629.

Ainda há, no entanto, um pequeno aumento de taxa de bits utilizados com o tamanho da mensagem. Isso não deveria ocorrer. Para mensagens muito grandes, o número de bits necessários a uma decodificação deveria tender a k.

Na Figura 3.6 pode-se observar o desempenho da distribuição Sóliton ideal.

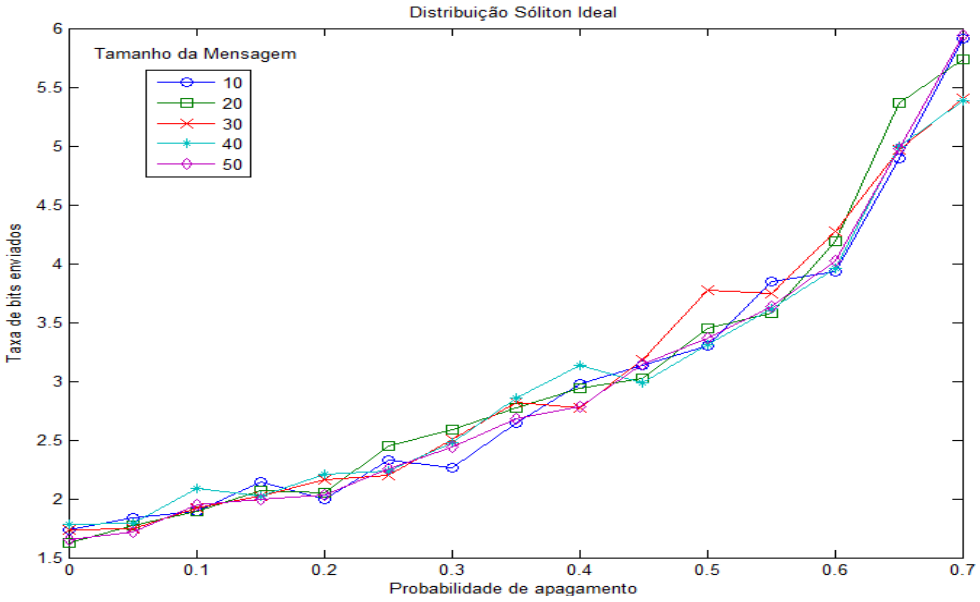

Figura 3.6 - Gráfico de taxa de bits enviados em relação à probabilidade de apagamento do canal para uma distribuição Sóliton Ideal.

3.4.4 – Sóliton Robusta

Embora o resultado da distribuição Sóliton ideal ainda seja ruim, ela serve de base para melhoras. Foi visto que o seu problema é determinar que o valor esperado do *ripple* fosse exatamente um. Dessa forma, variações em relação a isso fariam com que ele fosse zerado. A distribuição **Sóliton Robusta** (μ) modifica esse valor esperado com dois parâmetros. Dessa forma, pode-se aumentá-lo, porém, sem torná-lo muito grande. Assim, haverá uma grande probabilidade de que o *ripple* não desapareça.

Esta distribuição também foi definida por Luby [12]:

$\forall i: 0 < i \leq k, \qquad \mu(i) = (\rho(i) + \tau(i)) / \beta$

onde μ é a função Sóliton Robusta, ρ é a função Sóliton Ideal, β é uma constante de normalização igual a $\sum_i^k(\rho(i) + \tau(i))$, e τ é a função positiva definida por:

- $\tau(i) = S / ki \qquad \forall\, i: 0 < i < (k / S)$
- $\tau(i) = S \cdot \ln(S/\delta) / k \qquad \text{para } i = (k / S)$
- $\tau(i) = 0 \qquad \forall\, i > (k / S)$

Onde S representa o número médio de símbolos codificados de grau unitário e é igual a $c \cdot \sqrt{k} \cdot \ln(k/\delta)$, δ é uma constante que representa a chance de erro de leitura da mensagem e, por fim, 'c' é outra constante, entre zero e um.

Foi mostrado por Luby [12] que para uma boa escolha de 'c', o decodificador pode recuperar a mensagem original a partir de k.β símbolos codificados, com probabilidade de falha menor do que δ.

Na Figura 3.7 a distribuição μ está representada. Foram utilizados c = 0,2; δ = 0,1; e k = 1000.

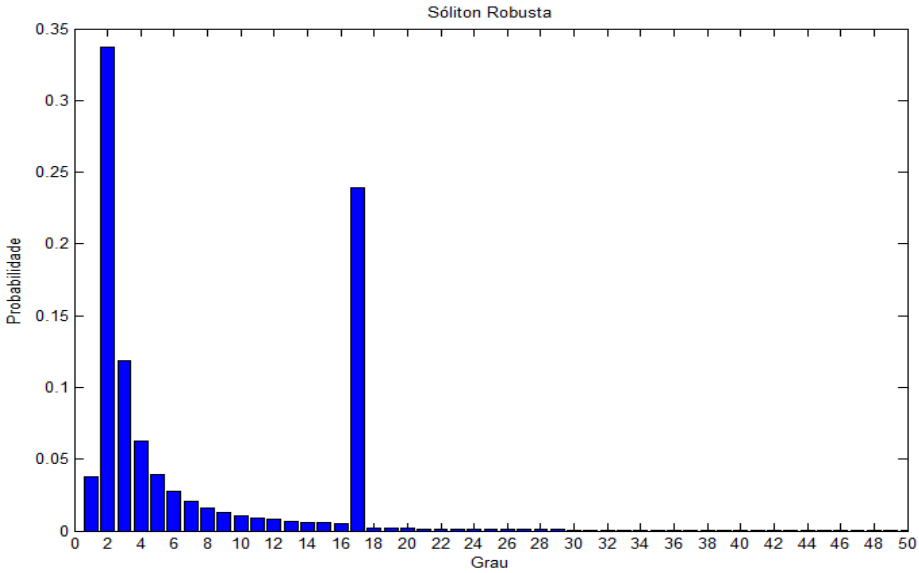

Figura 3.7 - Distribuição Sóliton Robusta.

Os fatos interessantes de serem ressaltados no gráfico são: o aumento da probabilidade de grau unitário e o pico no grau dezessete. Essas foram as soluções para os problemas que ocorreram no caso ideal. Aumentando-se o grau um, assegura-se um *ripple* maior. Já o pico em um grau relativamente alto serve para englobar mais bits. Uma quantidade grande de elementos de **X** com grau elevado faz com que, provavelmente, não haja bits da mensagem que não estejam incluídos em algum x_i.

O desempenho da Sóliton robusta é mostrado, então, na Figura 3.8:

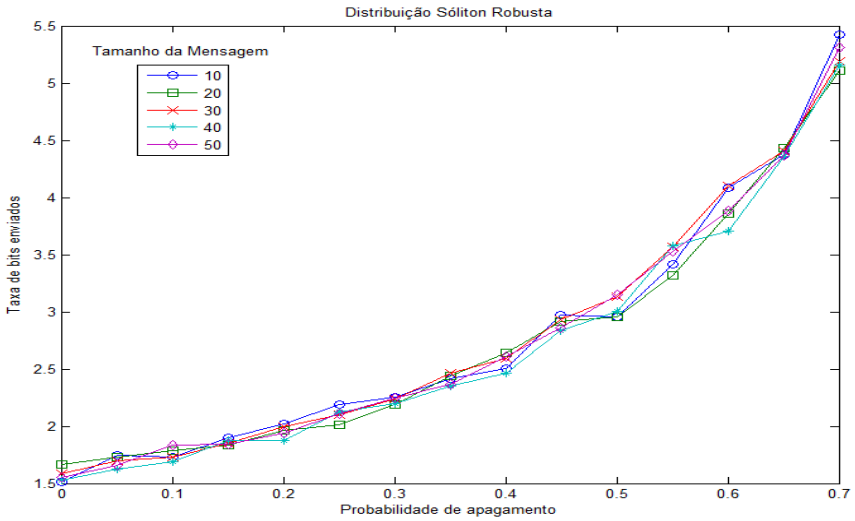

Figura 3.8 - Gráfico de taxa de bits enviados em relação à probabilidade de apagamento do canal para uma distribuição Sóliton Robusta.

Nota-se, agora, que a diferença devido ao tamanho da mensagem é quase imperceptível. Além disso, todas as taxas alcançadas foram menores do que as respectivas taxas das outras distribuições. Para uma mensagem de dez bits em um canal sem apagamento, em média, enviam-se 31,64 bits ao se utilizar a distribuição uniforme. Para a de grau unitário, 28,89. Para a Sóliton ideal, apenas 16,29. E, enfim, para a Sóliton Robusta, são necessários 15,10 bits. A taxa, portanto, é 1,510.

A comparação de desempenhos entre as distribuições pode ser vista a seguir. A Figura 3.9 mostra os desempenhos com mensagens de comprimento igual a dez. Já a Figura 3.10, expõe o caso que se usam cinquenta bits. Percebe-se que o uso da distribuição Sóliton Robusta é, realmente, o melhor. A diferença em relação às outras ainda aumenta quando se trabalha com mensagens de dimensões maiores.

Nas figuras, GU representa a distribuição de grau unitário, EQ, a equiprovável, SI, a sóliton ideal e SR, a sóliton robusta.

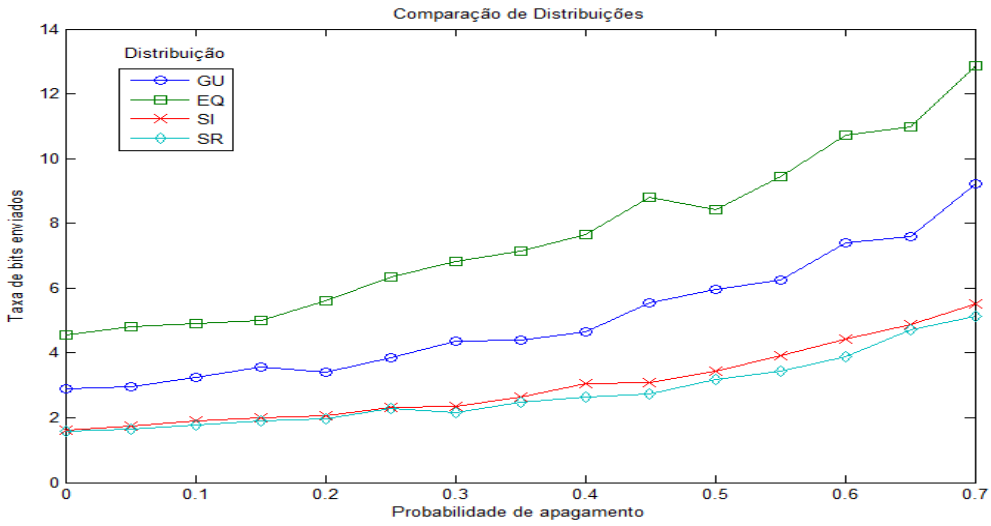

Figura 3.9 - Gráfico de taxa de bits enviados em relação à probabilidade de apagamento do canal para comparação de desempenho das distribuições, utilizando-se mensagens de comprimento de dez bits.

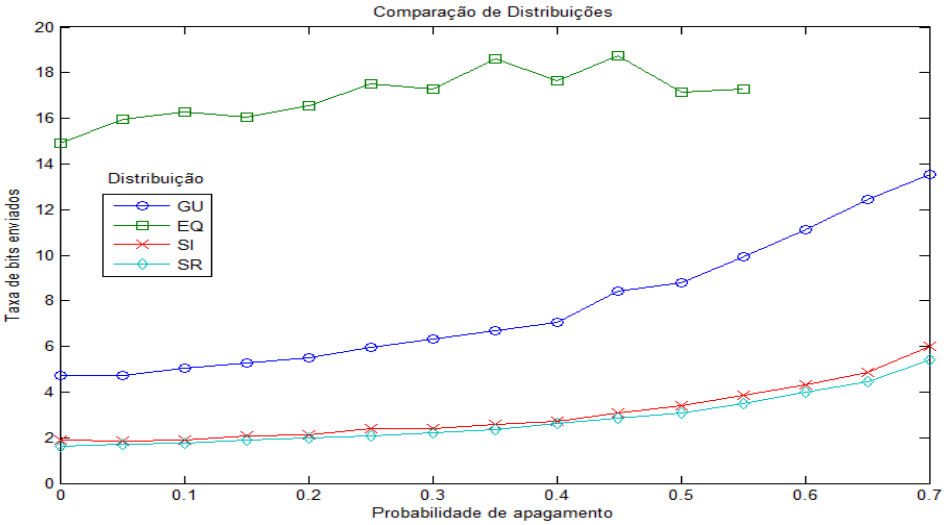

Figura 3.10 - Gráfico de taxa de bits enviados em relação à probabilidade de apagamento do canal para comparação de desempenho das distribuições, utilizando-se mensagens de comprimento de cinquenta bits.

Nesse capítulo foram analisados códigos fontanais. Em particular, códigos LT. Foram vistas diversas distribuições que podem ser utilizadas, para esse tipo de codificação. Estas foram comparadas, levando-se em conta seu desempenho para alguns tamanhos de mensagem e diferentes taxas de apagamento. Chega-se à conclusão que a melhor, dentre as distribuições, é a Sóliton Robusta. No próximo capítulo, esses resultados são aliados a novas idéias. São propostas, então, novas distribuições estatísticas para serem utilizadas da codificação. Seus desempenhos serão testados e comparados com os resultados anteriores.

Capítulo 4

Novas Propostas

4.1 – O uso dos graus altos da PDF

As análises do capítulo 3 nos mostraram a importância de um pequeno grau médio na mensagem codificada, afinal, existe a necessidade do *ripple* não ser zerado. Contudo, ela decorre do tipo de decodificação que é feito. Um elemento de grau unitário é necessário para que o processo seja deslanchado. Então, idealmente, cada bit decodificado faz com que outro seja coberto. Assim sendo, ocorre um processo em cadeia onde basta a decodificação de um elemento para tornar outro coberto. Para que isso seja possível, faz-se necessário um grau médio baixo na distribuição estatística.

Essa não é, no entanto, a melhor forma de fazer a decodificação. É possível utilizar o lado com alto grau da distribuição estatística. Para tal, procede-se como se fosse feita uma análise de duas paridades cobrindo bits diferentes. Na revista Eureka, há um problema que expõe o conceito por trás desse outro tipo de decodificação [16]:

- Cem pessoas são postas em uma fila e cada uma delas recebe um chapéu, que pode ser preto ou branco. Cada pessoa só consegue ver os chapéus das pessoas que estão a sua frente. É pedido que cada uma delas tente adivinhar a cor do seu chapéu. Qual o máximo número de acertos que se pode garantir, dado que as pessoas podem combinar uma estratégia antes de recebê-los.

A estratégia mais simples é fazer pares. Então, a pessoa que está atrás fala a cor do chapéu à sua frente. Assim, o segundo sempre acertará. Em seguida, pensa-se em fazer trios. A pessoa de trás diz, por exemplo, preto se os chapéus à sua frente são iguais, ou branco, se forem diferentes. Dessa forma, aquele que está no meio,

vendo a cor do chapéu à sua frente, saberá a cor do seu próprio. Por fim, a pessoa da frente também saberá.

Expandindo-se o raciocínio, utiliza-se o conceito de paridade. As cores serão faladas das pessoas de trás para as da frente. E a pessoa de trás fala branco, caso a quantidade de chapéus brancos a sua frente seja par e preto, caso contrário. A penúltima pessoa sabe a paridade da quantidade de chapéus brancos estritamente à sua frente. Sabe, também, a paridade da quantidade de chapéus brancos à sua frente, incluindo ela mesma, que foi informada pela última pessoa. Analisando os dois dados, ela acertará o seu chapéu. Com a pessoa à sua frente, ocorre o mesmo, e assim sucessivamente.

Para se perceber como isso pode ser utilizado na decodificação, basta substituir as cores por zeros e uns. Além disso, percebe-se que a paridade pode ser feita com operações XOR. Supondo, então, que se saiba o XOR de todos os bits da mensagem. Se for recebido o XOR de todos os bits exceto determinado m_i, é possível se descobrir m_i fazendo-se uma operação XOR entre esses dois dados. Assim sendo, vê-se que é possível utilizar graus altos também, desde que se faça uma decodificação mais inteligente.

Para facilitar a compreensão deste modo diferente de se decodificar, segue-se um exemplo:

M = {1; 0; 0; 1; 1}

Saída	Grau	Bits Utilizados	Valor
x1	3	m1; m2; m4	m1⊕ m2⊕ m4
x2	3	m1; m2; m5	m1⊕ m2⊕ m5
x3	3	m2; m3; m4	m2⊕ m3⊕ m4
x4	4	m1; m2; m3; m5	m1⊕ m2⊕ m3⊕ m5
x5	3	m1; m4; m5	m1⊕ m4⊕ m5

Tabela 4.1 - Exemplo de codificação LT com graus altos.

X = {0; 0; 1; 0; 1; 1}

Para decodificar a mensagem recebida, supõe-se, assim como no outro caso, que a matriz **G** é conhecida pelo receptor. Além disso, também é necessário o conhecimento do XOR entre todos os elementos de **M**. Este valor será denominado XOR total ou x_t. Esse bit pode ser transmitido da mesma forma que o receptor recebe a matriz **G**. Nesse caso, o seu valor é um. Assim sendo, a decodificação procede da seguinte forma: Procura-se um x_i que tenha grau igual ao comprimento total da mensagem menos um. Dessa forma, ele possui valor igual a $m_j \oplus x_t$. onde j é determinado pela matriz **G**. Com um bit da mensagem original conhecido, é possível aumentar um grau dos demais elementos de **X** que não 2o continham. Tal procedimento é repetido até que não haja mais bits a serem desvendados. No exemplo, a decodificação seria feita da seguinte forma, mostrada na :

1. Na primeira iteração, apenas x_4 está conectado a quatro símbolos de entrada. Ele apenas não está conectado ao m_4.
2. Faz-se, então, $m_4 = x_4 \oplus x_t$ e elimina-se, a seguir, o nó de verificação x_4.
3. Faz-se o XOR de todos os elementos de **X** que não possuam m_4 e lhes atribui um grau a mais. No caso, o único que se enquadra é o x_2. Dessa forma, cria-se um novo x_2 que tem grau quatro e contém m_4.
4. No inicio da segunda iteração, o segundo nó de verificação está ligado a quatro símbolos. Faz-se então $m_3 = x_2 \oplus x_t$ e elimina-se o nó de verificação.
5. Adiciona-se o valor de m_3 aos nós de verificação que não estão ligados a ele.
6. No inicio da terceira iteração, o primeiro nó de verificação não está ligado apenas a m_5. Faz-se então $m_5 = x_1 \oplus x_t$ e elimina-se o nó de verificação x_1.
7. Adiciona-se o de valor m_5 aos x_3.
8. No inicio da quarta iteração, escolhe-se o terceiro nó de verificação que não está ligado apenas a m_1. Faz-se então $m_1 = x_3 \oplus x_t$ e eliminando-o, então.
9. Por fim, resta apenas resolver m_2, utilizando x_5. Faz-se $x_5 = m_2 \oplus x_t$. então, a decodificação estará completa.

1-

X	Grau	Bits Utilizados	Valor
xt	5	m1; m2; m3; m4; m5	1
x1	3	m1; m2; m4	0
x2	3	m1; m2; m5	0
x3	3	m2; m3; m4	1
x4	4	m1; m2; m3; m5	0
x5	3	m1; m4; m5	1

M	Valor
m1	
m2	
m3	
m4	
m5	

2-

X	Grau	Bits Utilizados	Valor
xt	5	m1; m2; m3; m4; m5	1
x1	3	m1; m2; m4	0
x2	3	m1; m2; m5	0
x3	3	m2; m3; m4	1
~~x4~~	~~4~~	~~m1; m2; m3; m5~~	~~0~~
x5	3	m1; m4; m5	1

M	Valor
m1	
m2	
m3	
m4	1
m5	

3-

X	Grau	Bits Utilizados	Valor
xt	5	m1; m2; m3; m4; m5	1
x1	3	m1; m2; m4	0
x2	4	m1; m2; m4; m5	1
x3	3	m2; m3; m4	1
~~x4~~	~~4~~	~~m1; m2; m3; m5~~	~~0~~
x5	3	m1; m4; m5	1

M	Valor
m1	
m2	
m3	
m4	1
m5	

4-

X	Grau	Bits Utilizados	Valor
xt	5	m1; m2; m3; m4; m5	1
x1	3	m1; m2; m4	0
~~x2~~	~~4~~	~~m1; m2; m4; m5~~	~~1~~
x3	3	m2; m3; m4	1
~~x4~~	~~4~~	~~m1; m2; m3; m5~~	~~0~~
x5	3	m1; m4; m5	1

M	Valor
m1	
m2	
m3	0
m4	1
m5	

5-

X	Grau	Bits Utilizados	Valor
xt	5	m1; m2; m3; m4; m5	1
x1	4	m1; m2; m3; m4	0
~~x2~~	~~4~~	~~m1; m2; m4; m5~~	~~1~~
x3	3	m2; m3; m4	1
~~x4~~	~~4~~	~~m1; m2; m3; m5~~	~~0~~
x5	4	m1; m3; m4; m5	1

M	Valor
m1	
m2	
m3	0
m4	1
m5	

6-

X	Grau	Bits Utilizados	Valor
xt	5	m1; m2; m3; m4; m5	1
~~x1~~	~~4~~	~~m1; m2; m3; m4~~	~~0~~
~~x2~~	~~4~~	~~m1; m2; m4; m5~~	~~1~~
x3	3	m2; m3; m4	1
~~x4~~	~~4~~	~~m1; m2; m3; m5~~	~~0~~
x5	4	m1; m3; m4; m5	1

M	Valor
m1	
m2	
m3	0
m4	1
m5	1

7-

X	Grau	Bits Utilizados	Valor
xt	5	m1; m2; m3; m4; m5	1
~~x1~~	~~4~~	~~m1; m2; m3; m4~~	~~0~~
~~x2~~	~~4~~	~~m1; m2; m4; m5~~	~~1~~
x3	4	m2; m3; m4; m5	0
~~x4~~	~~4~~	~~m1; m2; m3; m5~~	~~0~~
x5	4	m1; m3; m4; m5	1

M	Valor
m1	
m2	
m3	0
m4	1
m5	1

8-

X	Grau	Bits Utilizados	Valor
xt	5	m1; m2; m3; m4; m5	1
~~x1~~	~~4~~	~~m1; m2; m3; m4~~	~~0~~
~~x2~~	~~4~~	~~m1; m2; m4; m5~~	~~1~~
~~x3~~	~~4~~	~~m2; m3; m4; m5~~	~~0~~
~~x4~~	~~4~~	~~m1; m2; m3; m5~~	~~0~~
x5	4	m1; m3; m4; m5	1

M	Valor
m1	1
m2	
m3	0
m4	1
m5	1

9-

X	Grau	Bits Utilizados	Valor
xt	5	m1; m2; m3; m4; m5	1
~~x1~~	~~4~~	~~m1; m2; m3; m4~~	~~0~~
~~x2~~	~~4~~	~~m1; m2; m4; m5~~	~~1~~
~~x3~~	~~4~~	~~m2; m3; m4; m5~~	~~1~~
~~x4~~	~~4~~	~~m1; m2; m3; m5~~	~~0~~
~~x5~~	~~4~~	~~m1; m3; m4; m5~~	~~1~~

M	Valor
m1	1
m2	0
m3	0
m4	1
m5	1

Figura 4.1 - Exemplo de decodificação de código LT com graus altos.

4.2 – Inversão da Distribuição de Grau Unitário

Para iniciar o estudo dessa nova proposta, será analisada a distribuição mais simples, que foi vista no capítulo anterior. Contudo, o objetivo, agora, é obterem-se graus altos. Assim sendo, é interessante inverter a distribuição previamente utilizada. Ou seja, a probabilidade de 100%, que antes era para graus iguais a um, agora incide sobre graus iguais ao tamanho da mensagem menos um. É importante ressaltar que, ao inverter a distribuição, são esperados resultados semelhantes aos obtidos com as técnicas mostradas no capítulo 3. Afinal, o mecanismo de decodificação tem a mesma base: A resolução de sistemas. A diferença é que, no caso tradicional, diminui-se o grau dos elementos para resolvê-los. Em contrapartida, na nova proposta, os graus devem ser aumentados para se chegar à resolução.

Na Figura 4.2, é visto o desempenho da distribuição de **Grau Unitário Invertida**. Como previsto, os resultados foram muito semelhantes aos conseguidos com a distribuição não invertida. Para uma mensagem de dez bits em um canal de apagamento nulo, a distribuição não invertida envia 28,89 bits enquanto a invertida envia 26,10.

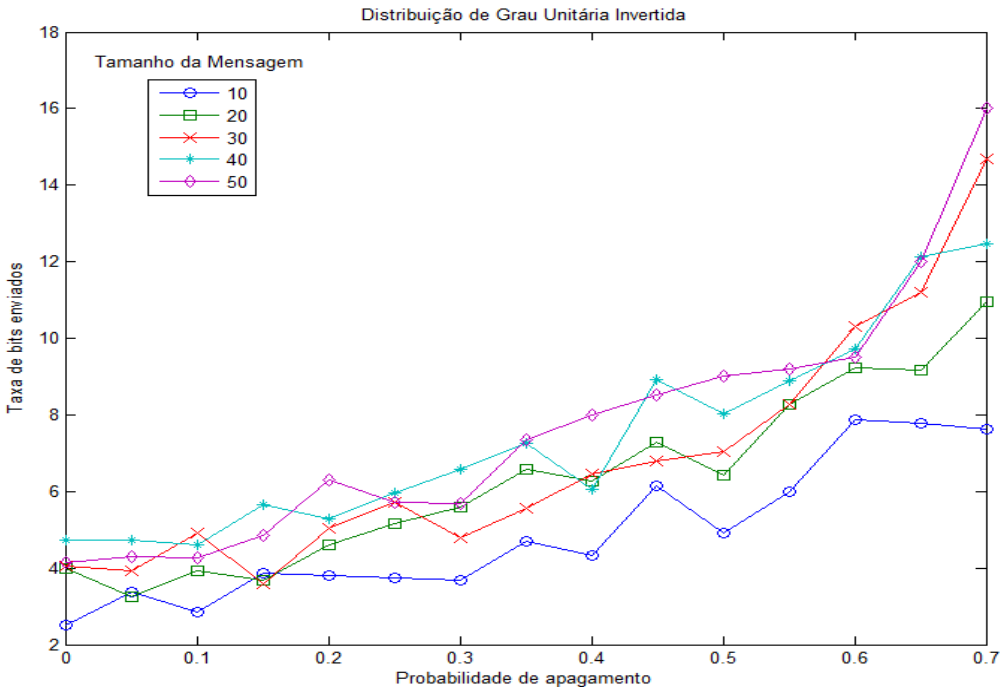

Figura 4.2 - Gráfico de taxa de bits enviados em relação à probabilidade de apagamento do canal para uma distribuição de grau unitário invertida.

4.3 – Inversão da Sóliton Ideal

A distribuição equiprovável tem péssimos rendimentos, sobretudo, com mensagens grandes. Portanto, não é necessária sua análise nessa parte do projeto. Dessa forma, o próximo passo é a análise da distribuição Sóliton Ideal Invertida. Da mesma forma que a distribuição de grau unitário, é necessário fazer com que os graus baixos tornem-se os altos e vice-versa. Com essa inversão, também se esperam resultados semelhantes aos obtidos no capítulo 3. A nova distribuição produz, realmente, taxas muito próximas àquelas. Pode-se observar o seu desempenho na Figura 4.3. Enquanto a invertida envia 15,70 bits, para a decodificação de uma mensagem de dez bits sem apagamento, a normal envia 16,29.

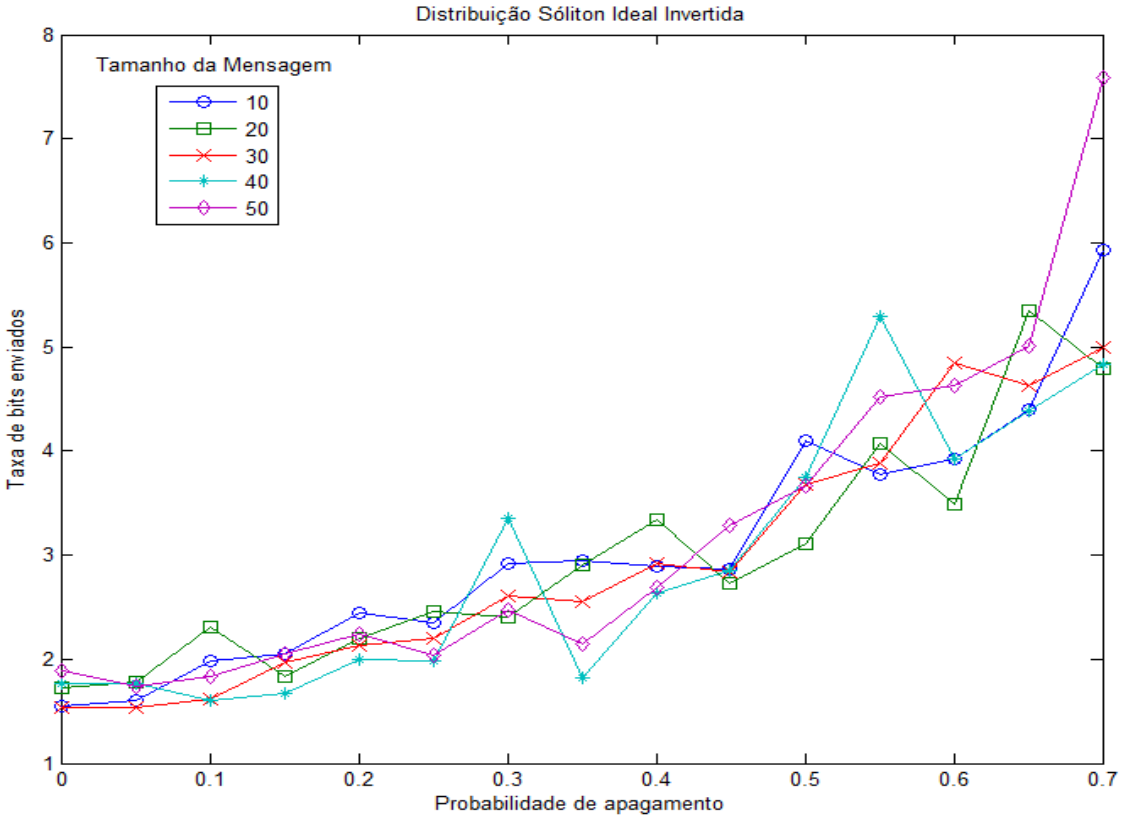

Figura 4.3 - Gráfico de taxa de bits enviados em relação à probabilidade de apagamento do canal para uma distribuição Sóliton ideal invertida.

4.4 – Inversão da Sóliton Robusta

Para finalizar a parte de inversão de distribuições, utilizamos a Sóliton Robusta. Esta produziu os melhores resultados para a codificação. A fim de verificar se os dados serão mantidos, como previsto, mais uma inversão de distribuição é feita.

É possível notar, através do gráfico da Figura 4.4, que esse caso, também, possui uma eficiência próxima à obtida através da decodificação tradicional. Parra uma mensagem de comprimento igual a dez bits, sem apagamento, a distribuição normal envia 15,10 bits. A distribuição invertida utiliza 15,40.

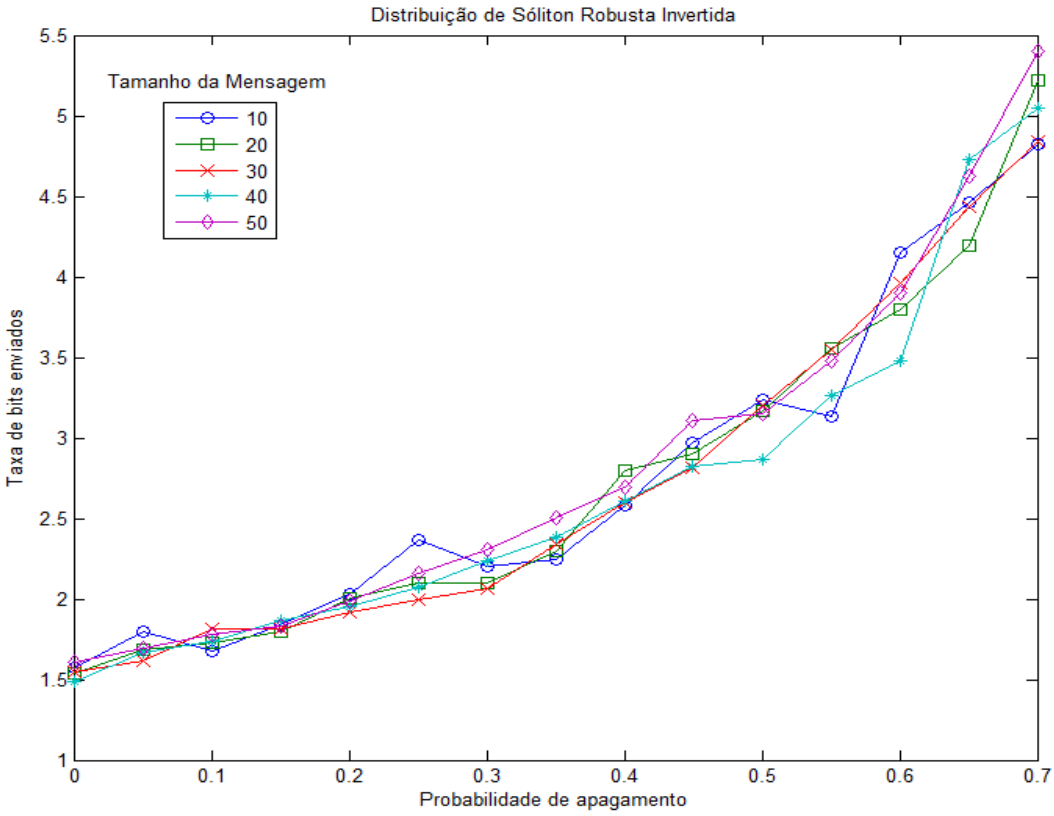

Figura 4.4 - Gráfico de taxa de bits enviados em relação à probabilidade de apagamento do canal para uma distribuição Sóliton robusta invertida.

4.5 – Exclusão do XOR total

Ao analisar como é feita a decodificação, fica óbvio o papel crucial que o XOR total possui. A fim de se mostrar sua importância para a taxa, fez-se uma análise onde não era garantida a presença do XOR total no receptor. Dessa forma, ele deve esperar que chegue um bit de grau igual ao tamanho da mensagem, para começar a decodificar. No gráfico da Figura 4.5, contrastam-se os desempenhos de uma distribuição Sóliton Ideal e uma Robusta para os casos com e sem XOR total. Foram utilizadas mensagens de dimensão igual a dez bits. Percebe-se que a ausência desse elemento produz a perda de eficiência. No gráfico, SI é a distribuição Sóliton Ideal, enquanto a SR é distribuição Sóliton Robusta.

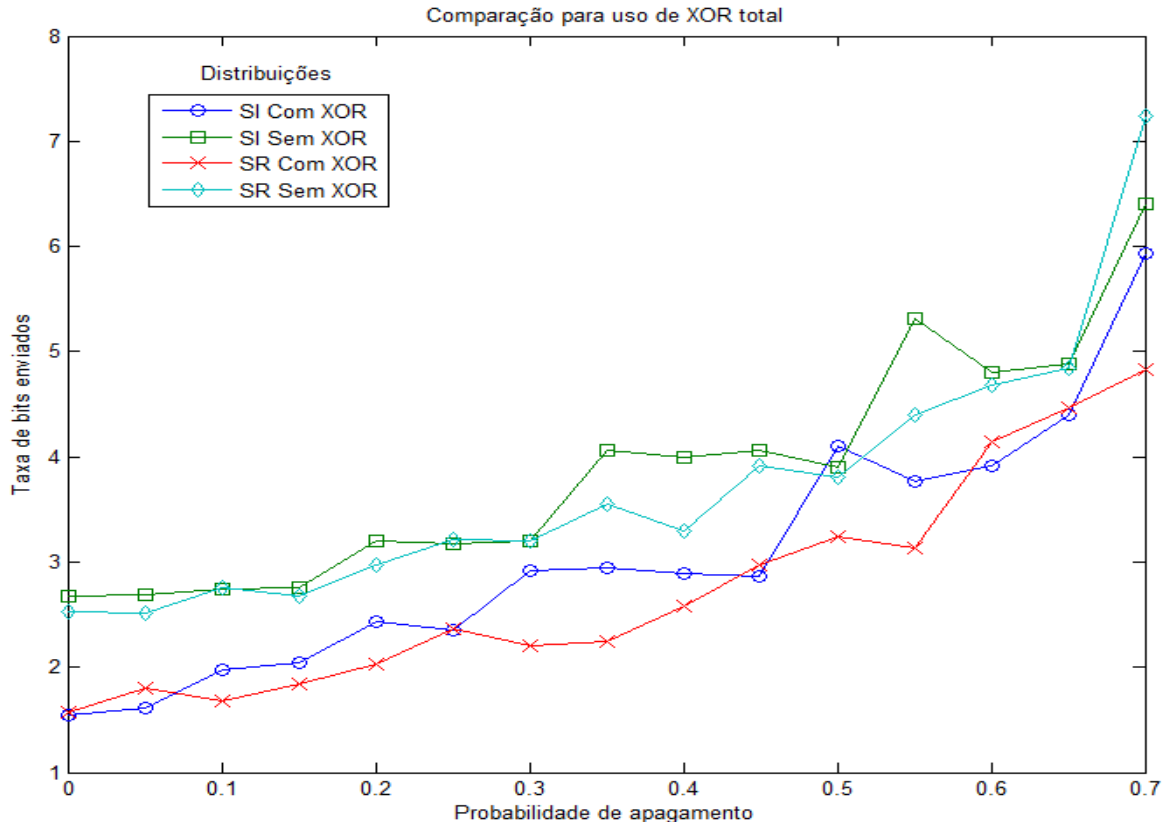

Figura 4.5 - Comparação dedesempenho de distribuições com o uso do XOR total e sem o mesmo.

4.6 – Uso de ambos os lados da PDF

A proposta, finalmente, para a melhora de desempenho, é a união dos dois métodos. Ou seja, decodifica-se através da diminuição dos graus das variáveis de controle. Em paralelo, utiliza-se a solução que utiliza o aumento de graus. Cada bit da mensagem que for descoberto por um deles, pode ser utilizado pelo outro.

Naturalmente, a decodificação sendo feita de duas formas paralelamente, acarreta em uma menor velocidade de processamento. Contudo, essa diminuição não é tão expressiva, uma vez que ambas as decodificações continuam sendo baseadas em operações XOR, que são de rápido processamento.

Não adianta, entretanto, utilizar uma das distribuições projetadas, especificamente para um desses métodos. Caso isso ocorra, uma das técnicas de decodificação será inútil. A única consequência será a diminuição da velocidade de processamento, o que não é interessante. Por conseguinte, deve-se, ainda, criar novas distribuições que aproveitem aspectos abordados pelos dois métodos.

A implementação dessa decodificação é, realmente, apenas a união das duas anteriores. A primeira parte decodifica o bit de grau um, que é excluído da tabela do primeiro método e é incluído na do segundo. Em seguida tenta-se descobrir um elemento de **X** que não seja ligado a apenas um elemento. Ele é decodificado, utilizado e enviado para a outra parte utilizá-lo.

A seguir, são apresentadas duas propostas de distribuições para usar com essa decodificação mais completa.

4.6.1 – Reflexão da Sóliton Ideal

As primeiras propostas que vêm à mente são as uniões das distribuições utilizadas para cada método exposto. É exatamente isso que será testado. A Sóliton Ideal foi a distribuição que teve o segundo melhor desempenho entra as observadas nesse projeto. Logo, para se utilizar tanto os graus altos como os baixos, ela foi somada com sua correspondente invertida e, em seguida, dividida por dois. Dessa forma, garante-se, ainda, a soma das probabilidades igual a um. Além disso, os dois lados da PDF são explorados, como pode ser visto na Figura 4.6.

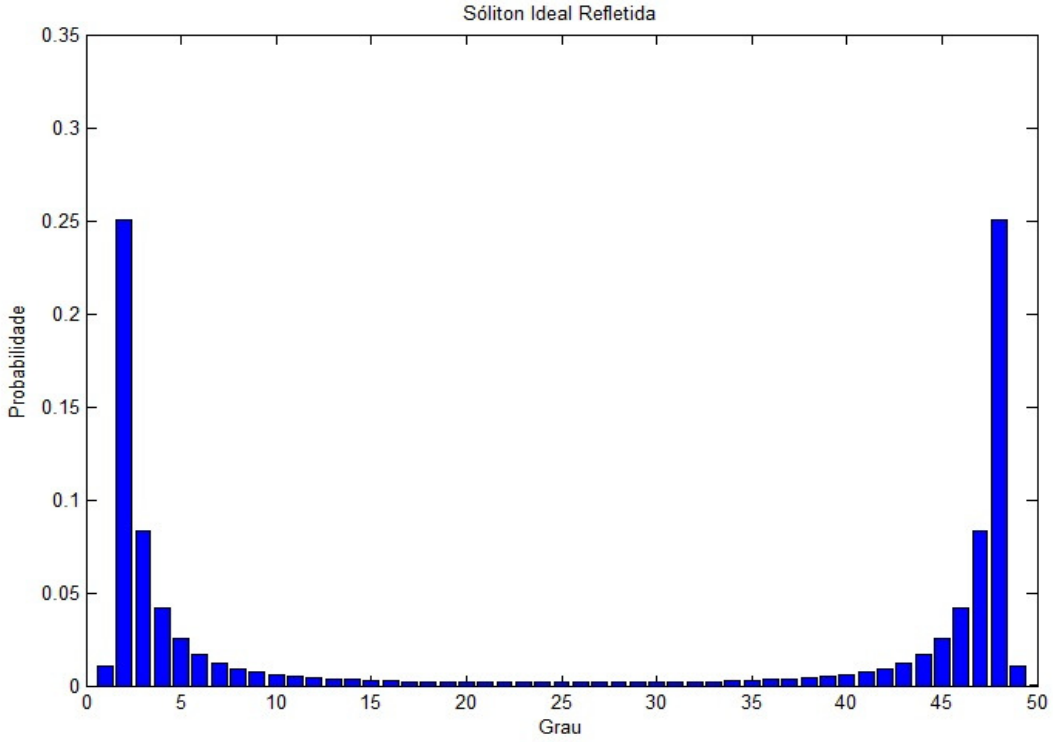

Figura 4.6 - Distribuição Sóliton Ideal Refletida

Já na Figura 4.7, pode ser observado o desempenho da distribuição.

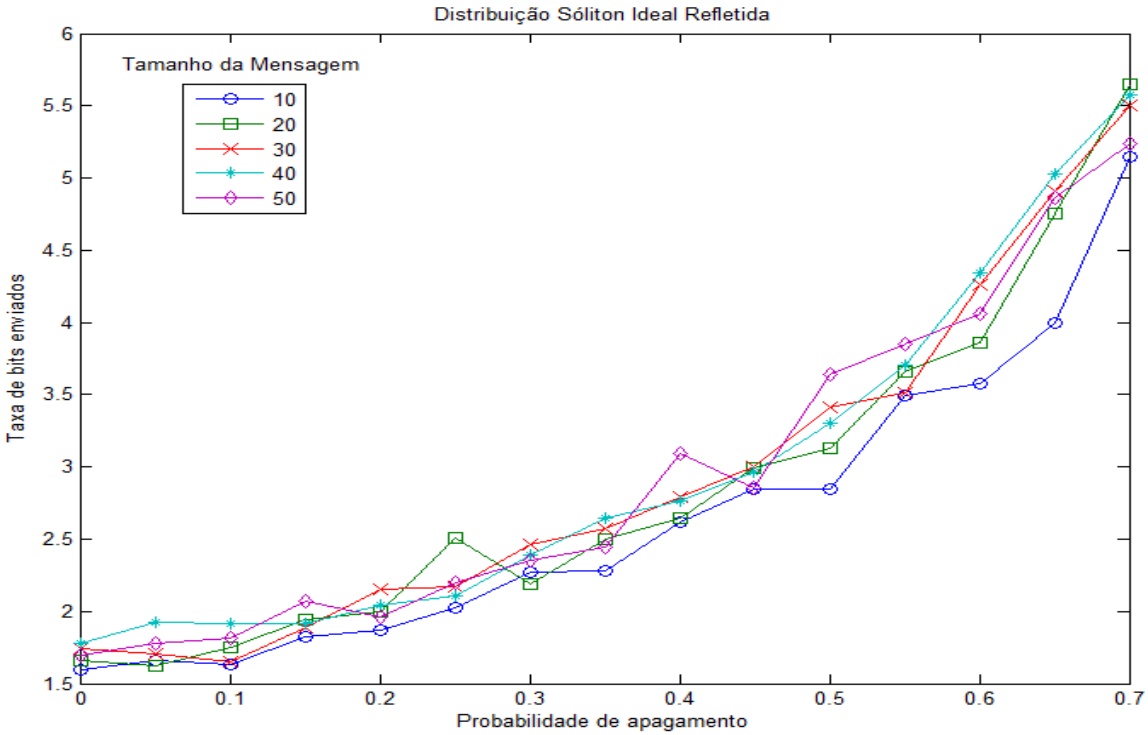

Figura 4.7 - Gráfico de taxa de bits enviados em relação à probabilidade de apagamento do canal para uma distribuição Sóliton ideal refletida.

Os resultados obtidos seguem o mesmo padrão dos anteriores, com a similaridade entre os testes feitos com mudança de tamanho da mensagem. A comparação para melhor análise será feita após ser vista, também, a distribuição sóliton robusta refletida.

4.6.2 – Reflexão da Sóliton Robusta

A melhor distribuição, nesse projeto, em questão de desempenho, foi a Sóliton Robusta. Consequentemente, a sua reflexão também deve ser testada. A distribuição, após ser somada com sua inversa e dividida por dois, assume a forma exposta na Figura 4.8.

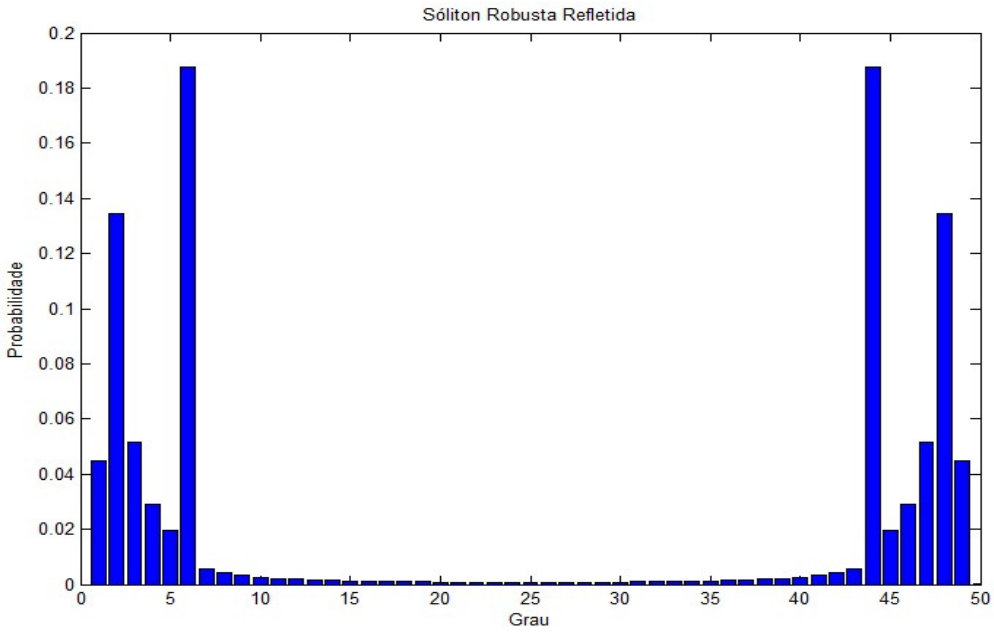

Figura 4.8 - Distribuição Sóliton Robusta Refletida

Por fim, podemos ver seu desempenho da Figura 4.9.

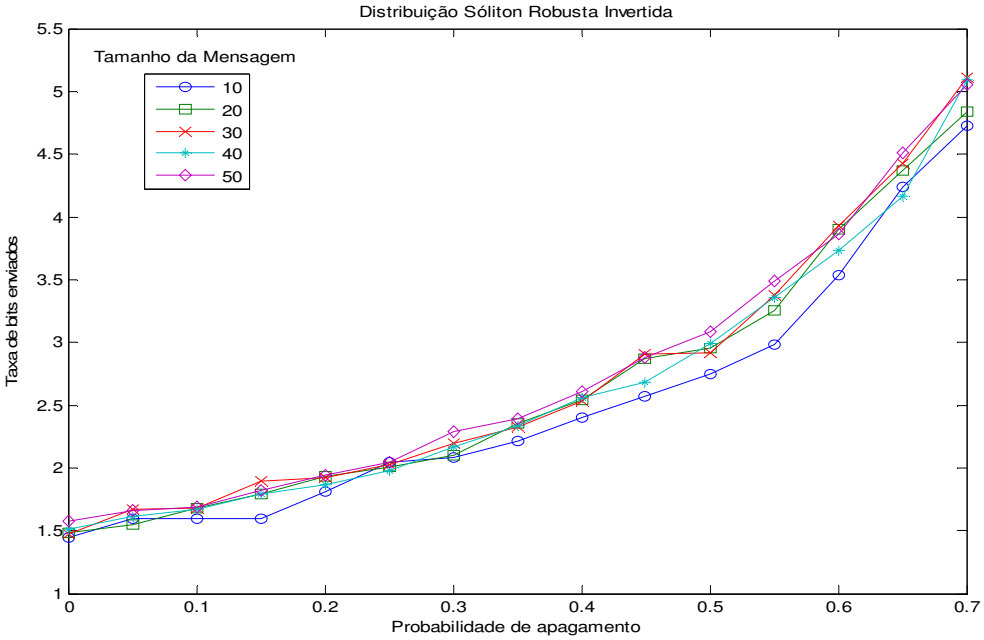

Figura 4.9 - Gráfico de taxa de bits enviados em relação à probabilidade de apagamento do canal para uma distribuição Sóliton robusta refletida.

Da mesma forma que na distribuição sóliton ideal, a similaridade entre as curvas obtidas com diferentes tamanhos de mensagem se mantém. No item seguinte é feita a análise de resultados

4.7 – Análise de resultados

Finalmente, pode ser feita a comparação entre os resultados obtidos quando são usadas as novas propostas e os do capítulo 3. Para tal, serão usados gráficos comparativos. Serão feitas análises para as distribuições Sóliton Ideal e Sóliton Robusta. Serão analisados seus desempenhos da forma tradicional, invertida e refletida. As análises serão feitas utilizando-se mensagens de dez bits de comprimento.

Um detalhe a se observar é que essa comparação é um pouco desfavorável ao método normal. Como o XOR total tem que ser enviado sem erros, ele utilizará um número alto de bits. Dessa forma, seu envio acarretaria em um aumento relevante nas

taxas das decodificações propostas nesse trabalho. Entretanto, para mensagens de tamanhos maiores, esse acréscimo não seria tão importante.

Na Figura 4.10, vê-se a comparação da Sóliton Ideal. É possível observar um desempenho ligeiramente melhor para a decodificação combinada do que com as outras duas separadas.

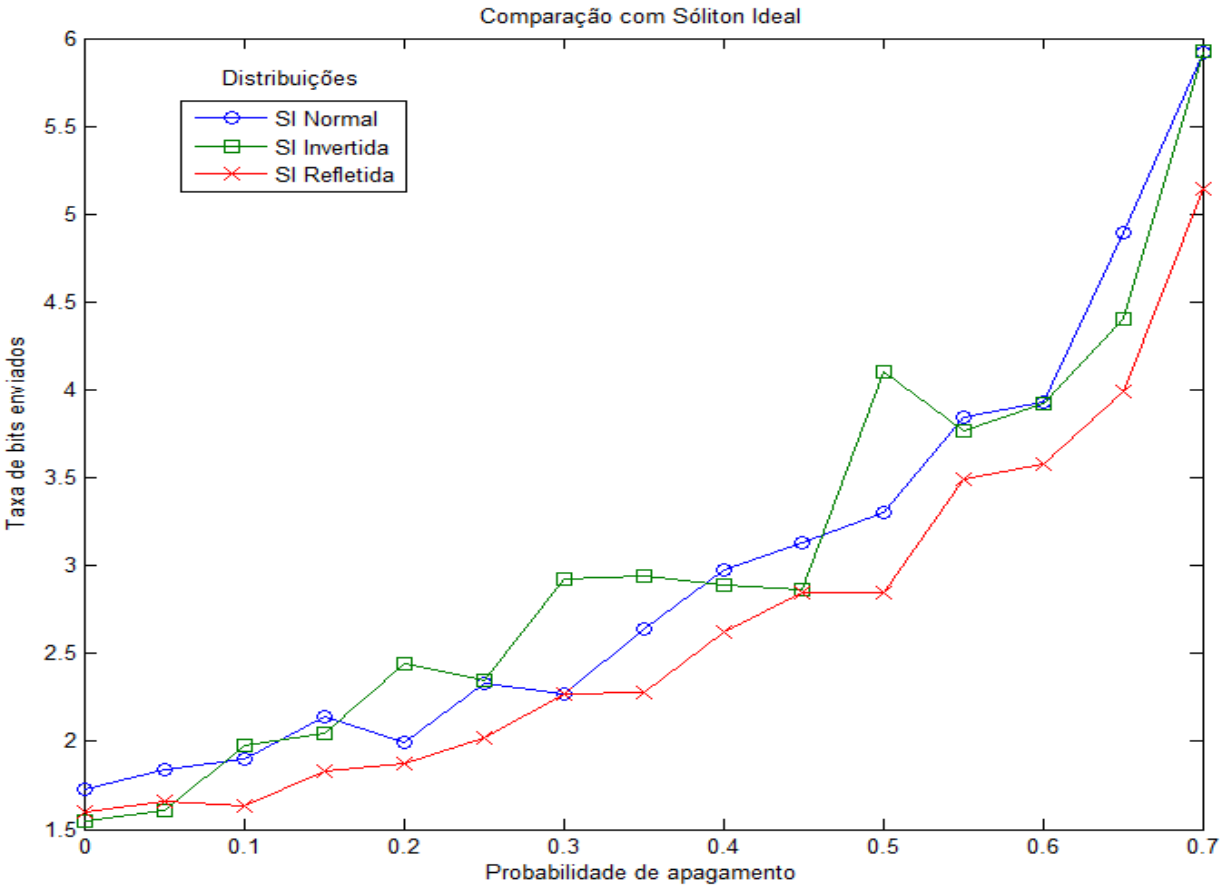

Figura 4.10 - Comparação dos desempenhos das decodificações utilizando a distribuição Sóliton Ideal

Por fim, na Figura 4.11, vê-se a comparação de desempenho quando se utiliza a distribuição Sóliton Robusta. Da mesma forma, é possível notar que a decodificação com a união das duas técnicas é a que tem o melhor desempenho.

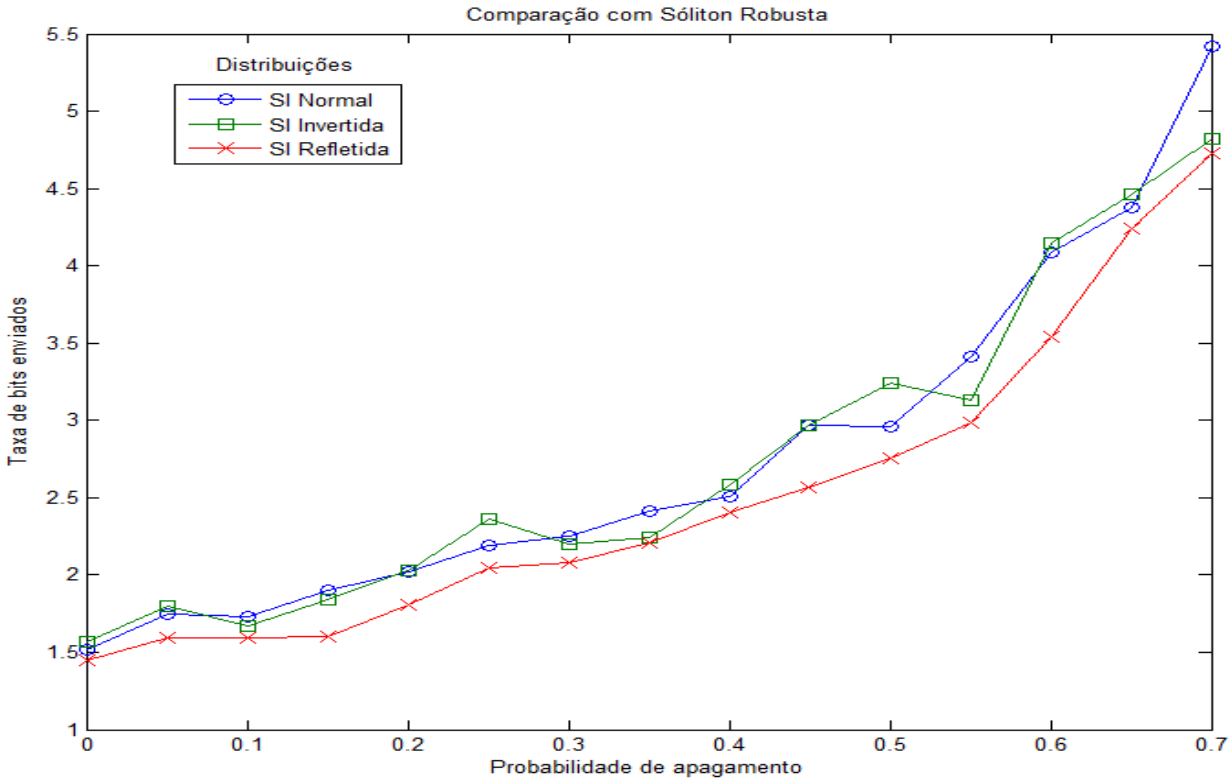

Figura 4.11 - Comparação dos desempenhos das decodificações utilizando a distribuição Sóliton Robusta

Através das propostas feitas nesse capítulo buscou-se fugir do método mais tradicional de decodificação para códigos LT. Foram obtidos resultados melhores do que os tradicionais. No capítulo seguinte, faz-se a conclusão e a verificação dos objetivos do projeto.

Capítulo 5

Conclusão

5.1 – Contribuições

O projeto começou com um estudo de Teoria da Informação, explorando, principalmente a parte relativa à codificação de canal. Foram apresentados diversos conceitos importantes, assim como exemplos de códigos.

Em seguida, viu-se o conceito de codificação de canal com taxa indefinida (*rateless*). Um caso prático é o código LT. Foi feita a sua implementação em Matlab, que era o primeiro objetivo do projeto. Após se verificar o correto funcionamento do algoritmo, testaram-se as distribuições: Grau Unitário; Equiprovável; Sóliton Ideal; Sóliton Robusta. Completou-se, então, o segundo objetivo.

No capítulo seguinte, foi proposto o uso dos graus altos da PDF de entrada. Dessa forma, foram feitos novos testes para as distribuições invertidas. Em seguida, para se obter uma melhora, as duas propostas, a tradicional e a nova, são unidas. Logo, são feitos outros testes para as mesmas distribuições, mas, agora, refletidas. Os resultados são, por fim, comparados e avaliados, completando-se, portanto, os três últimos objetivos do trabalho.

5.2 – Possíveis Continuações

Este projeto visa expor uma nova forma de se decodificar códigos LT. Abre-se a perspectiva de um estudo de distribuições que explorem essa decodificação.

Há, também, técnicas de melhora de desempenho que podem ser combinadas com a exposta nesse trabalho. Um exemplo é o *lengthening* exposto em "Improving the Performance of LT Codes" [17]. Esse método consiste em adicionar, ao final da mensagem a ser transmitida, um grupo de bits conhecidos pelo transmissor e pelo receptor. Dessa forma, são obtidas mensagens maiores e taxas menores.

Bibliografia

[1] GUIZZO, Erico M. – "The Essential Message: Claude Shannon and the Making of Information Theory" – Program in Writing and Humanistic Studies – MIT – Setembro, 2003.

[2] SHANNON, Claude E. – "A Mathematical Theory of Communication" – The Bell System Technical Journal, Vol. 27, pp. 379–423, 623–656, Julho, Outubro, 1948.

[3] http://www.nytimes.com/2001/02/27/nyregion/claude-shannon-mathematician-dies-at-84.html - acessado em 20/07/2010.

[4] LARS, Lundheim – "On Shannon and 'Shannon's formula'" – Telektronikk, vol. 98, no. 1, pp. 20-29, ISSN 0085-7130, published by Telenor – 2002.

[5] COVER, Thomas M.; THOMAS, Joy A. – "Elements of Information Theory" – New York, A Wiley –Interscience Publication 1991 – 2nd Edition.

[6] SHANNON, Claude E. – "Communication in the Presence of Noise" - Proceedings Of The Ire, Vol.37, No.1, pp. 10–21, Janeiro, 1949.

[7] HAYKIN, Simon – "Sistemas de Comunicação Analógicos e Digitais" – 4ª edição – Porto Alegre, Bookman 2004.

[8] ELIAS, Peter – "Coding for two noisy channels" – Information Theory, Third London Symposium, p. 61-76, 1955.

[9] BYRES, John W.; LUBY, Michael; MITZENMACHER, Michael – "A Digital Fountain Approach to Asynchronous Reliable Multicast" – IEEE J. on Selected Areas in Communications, Special Issue on Network Support for Multicast Communications, Vol. 20:pag. 1528-1540, Agosto 2002.

[10] PAIBA, Franklin S. – "Códigos Fontanais Bidimensionais para Canais com Apagamento" – Dissertação de Mestrado – PUC – Julho, 2008.

[11] MITZENMACHER, Michael – "Digital Fountains: A Survey and Look Forward" – IEEE Information Theory Workshop, p. 24-29, Outubro 2004.

[12] LUBY, Michael – "LT codes" – Proc. of the 43rd Annual IEEE Symposium on Foundation of Comp. Sc., p. 271-280, Novembro 2002.

[13] MACKAY, David – "Capacity Approaching Codes Design And Implementation" - IEE Proc.- Community, Vol. 152, No. 6, p.1062-1068 – Dezembro, 2005.

[14] KHISTI, Ashish – "Tornado Codes and Luby Transform Codes" – October, 2003 – http://web.mit.edu/6.454/www/www_fall_2003/khisti/ – Acessado em 17/08/2010.

[15] HYYTIÂ, Esa; TIRRONEN, Tuomas; VIRTAMO, Jorma – "Optimizing the Degree Distribution of LT Codes with an Importance Sampling Approach" –2006 – RESIM 2006, 6th International Workshop on Rare Event Simulation, Bamberg, Alemanha.

[16] NASCIMENTO JR, Einstien do; FEITOSA, Samuel B. – "Par ou ímpar? Eis a questão" – Eureka, nº 31, p. 13 – 14 – 2010 .

[17] FINAMORE, Weiler A.; RAMOS, Marcelo C. – "Improving the Performance of LT Codes"