AN ANALYSIS ON LUBY TRANSFORM ENCODING-DECODING ALGORITHMS AND PROPOSALS FOR MORE EFFICIENT PRACTICAL USAGE

A THESIS SUBMITTED TO

THE GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES

OF

MIDDLE EAST TECHNICAL UNIVERSITY

MEHMET BÜLBÜLDERE

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS

FOR

THE DEGREE OF MASTER OF SCIENCE

IN

ELECTRICAL AND ELECTRONICS ENGINEERING

JANUARY 2015

Approval of the thesis:

AN ANALYSIS ON LUBY TRANSFORM ENCODING-DECODING ALGORITHMS AND PROPOSALS FOR MORE EFFICIENT PRACTICAL USAGE

submitted by **MEHMET BÜLBÜLDERE** in partial fulfillment of the requirements for the degree of **Master of Science in Electrical and Electronics Engineering Department, Middle East Technical University** by,

Prof. Dr. Gülbin Dural Ünver
Dean, Graduate School of **Natural and Applied Sciences** _____

Prof. Dr. Gönül Turhan Sayan
Head of Department, **Electrical and Electronics Engineering** _____

Assoc. Prof. Dr. Melek Diker Yücel
Supervisor, **Electrical and Electronics Engineering Dept.** _____

Examining Committee Members:

Prof. Dr. Yalçın Tanık
Electrical and Electronics Engineering Dept., METU _____

Assoc. Prof. Dr. Melek Diker Yücel
Electrical and Electronics Engineering Dept., METU _____

Prof. Dr. Ali Özgür Yılmaz
Electrical and Electronics Engineering Dept., METU _____

Assoc. Prof. Dr. Çağatay Candan
Electrical and Electronics Engineering Dept., METU _____

Sıdıka Bengür, M.Sc.
Manager of HBT-PHSMM, ASELSAN Inc. _____

Date: January 22, 2015

I hereby declare that all information in this document has been obtained and presented in accordance with academic rules and ethical conduct. I also declare that, as required by these rules and conduct, I have fully cited and referenced all material and results that are not original to this work.

Name, Last Name : MEHMET BÜLBÜLDERE

Signature :

ABSTRACT

AN ANALYSIS ON LUBY TRANSFORM ENCODING-DECODING ALGORITHMS AND PROPOSALS FOR MORE EFFICIENT PRACTICAL USAGE

Bülbüldere, Mehmet

M.S., Department of Electrical and Electronics Engineering

Supervisor : Assoc. Prof. Dr. Melek Diker Yücel

January 2015, 88 pages

We study the encoding and decoding algorithms of Luby Transform Codes and analyze the performance of these codes. Luby Transform Coding technique is one of the classes of rateless codes. Depending on the use scenario, higher data rates can be obtained by using LT codes instead of using only conventional fixed rate codes. Segmentation of data to be transmitted and channel conditions have major effects on the speed of transmission of whole data. Throughput, data rate, decoding success ratio results, which are obtained through the LT code simulator and conventional fixed rate code simulator constructed in MATLAB during this thesis work, are analyzed according to the several factors such as signal to noise ratio (SNR) of received signals, symbol length, number of input symbols, packet erasure probability for both cases with and without LT codes. In addition, different distributions of encoding symbol degrees are studied for LT codes since degree distribution is another significant factor affecting the obtained throughput. The cases with and without LT codes are compared to each other for both point-to-point and point-to-multipoint transmission cases. Furthermore, effect of using error correction codes is analyzed for different received SNR values. Since their structures are open to be modified flexibly, the simulator constructed for this thesis can be used for numerous different practical scenarios while deciding on the optimum parameters to obtain aimed decoding success ratios, data rates and throughputs.

Keywords: Luby Transform, LT codes, LT encoder, LT decoder, rateless codes, fountain codes, throughput, point-to-multipoint transmission

ÖZ

LUBY TRANSFORM KODLAMA-ÇÖZÜMLEME ALGORİTMALARININ ANALİZİ VE DAHA VERİMLİ PRATİK KULLANIMLARI İÇİN ÖNERİLER

Bülbüldere, Mehmet

Yüksek Lisans, Elektrik ve Elektronik Mühendisliği Bölümü

Tez Yöneticisi : Doç. Dr. Melek Diker Yücel

Ocak 2015, 88 sayfa

Bu araştırmada, Luby Transform kodlarının kodlama ve çözümleme algoritmaları üzerine çalışılıp bu kodların performansları incelendi. Luby Transform kodlama tekniği, oransız kodların bir çeşididir. Kullanım senaryosuna bağlı olarak, sadece alışılagelmiş sabit oranlı kodları kullanmak yerine bu kodlarla birlikte LT kodlar kullanılırsa daha yüksek veri aktarım hızları elde edilebilir. Gönderilecek verinin bölütlenmesi (segmentation) ve kanal koşulları, tüm verinin aktarım hızında belirleyici etkiye sahiptir. Bu tez çalışması sırasında MATLAB aracılığıyla oluşturulan LT kod ve alışılagelmiş sabit oranlı kod benzeteçleri (simulator) ile elde edilen veri aktarımı (throughput), veri aktarım hızı ve kod çözümleme başarı oranı ile ilgili sonuçlar, alınan sinyal gücünün gürültü gücüne oranı (Signal to Noise Ratio-SNR), sembol uzunluğu, girdi (input) paketi sayısı ve paket silinme olasılıklarına göre hem LT kodlar hem de alışılageldik sabit oranlı kodlar için incelenmiştir. Ayrıca, elde edilen veri aktarım hızını etkileyen önemli bir etken olduğu için kodlama paketlerinin derece dağılımları üzerine çalışma yapılmıştır. LT kodlar ve alışılagelmiş sabit oranlı kodlar, hem bir vericiden bir alıcıya hem de bir vericiden birçok alıcıya gönderim yapılan durumlar için birbirleri ile kıyaslanmıştır. Ek olarak, farklı SNR değerleri için hata düzeltici kodların kullanımının etkisi incelenmiştir. Bu tez sırasında oluşturulan benzeteç esnek bir yapıya sahip olduğu için, birçok farklı gerçekçi senaryoya uyarlanarak hedeflenen kod çözümleme başarı oranları, veri aktarım hızları ve veri hacimleri için ilgili değişkenlerin en uygun değerlerinin bulunmasında kullanılabilir.

Anahtar Kelimeler: Luby Transform, LT kodlar, LT kodlayıcı, LT çözümleyici, oransız kodlar, pınar (fountain) kodları, veri aktarımı, tek-noktadan-çok-noktaya iletim

To my beloved family

ACKNOWLEDGMENTS

Firstly, I would like to give my honest appreciation to my supervisor Assoc. Prof. Dr. Melek Diker Yücel for her patience and motivating attitude. I could not have made this research without her invaluable guidance.

I would also like to thank my mother for her sincere and constant encouragement for this thesis.

I am thankful to my friends and colleagues. They always showed their trust in me. Their assistance means a lot to me.

Finally, I would like to indicate that I will always remain grateful to my company ASELSAN for the support during my thesis work.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

3G	3rd Generation
ARQ	Automatic Repeat Request
AWGN	Additive White Gaussian Noise
BPSK	Binary Phase Shift Keying
C4FM	Constant Envelope 4-Level Frequency Modulation
CRC	Cyclic Redundancy Check
LT	Luby Transform
LTE	Long Term Evolution
M2M	Machine-to-Machine
PSK	Phase-Shift Keying
QAM	Quadrature Amplitude Modulation
RX	Receiver
TX	Transmitter

CHAPTER 1

INTRODUCTION

As practical applications requiring data transmission become more prevalent and the need for such applications increase, two criteria gain more significance: to make the data transmission faster and to reliably decode the data even at low SNR values. Channel coding used for error detection and correction serves only one of these purposes; the redundancy added by coding increases reliability but it decreases the rate of information transmission as well.

Several different coding techniques, which have conventionally been used for years, have some constant coding rate. This rate is calculated by dividing the number of information bits (or symbols) to the number of channel input bits (or symbols). If k information bits are transmitted through the channel by using $n > k$ bits, where $n - k$ redundant bits are added for error correction and/or detection; the corresponding coding rate is k/n **[Wesolowski, 2009]**. For conventional coding techniques such as CRC, convolutional codes etc., coding rate is kept fixed during the transmission of data.

On the other hand, for rateless codes, which are also known as fountain codes, the coding rate is not constant, but it changes during the transmission of data. The encoding algorithm of rateless codes can in principle produce an infinite number of message packets **[Tirronen, Virtamo, 2006]**. An important advantage is that rateless codes can meet the reliability requirement without feedback channels.

Rateless codes are near-optimal erasure correcting codes **[Luby, 2002], [MacKay, 2005]**. Due to the efficient encoding and decoding algorithms of rateless codes, k source symbols can be recovered by using k' encoding symbols with high probability, where k' is slightly larger than k **[MacKay, 2005]**.

Rateless codes must be assessed as application layer coding techniques used with some other error correcting and/or detecting codes since they cannot correct or detect errors without the assist of conventional methods such as Cyclic Redundancy Check (CRC), convolutional codes, etc. **[Botha, 2008]**. The first practical realization of rateless codes, called Luby Transform codes **[Luby, 2002]**, were published in 2002 by Michael Luby.

In Luby Transform (LT) codes, encoding symbols are created by XOR'ing the input symbols according to a pre-assigned degree distribution and neighbor list. If the degree is one, there will be only one neighbor input symbol and the corresponding encoding symbol will be a copy of this input symbol. Degree disribution is one of the key points to continue the decoding process properly and to obtain a high throughput. In Luby's **[Luby, 2002]** and MacKay's **[MacKay, 2005]** papers, firstly All-At-Once and Ideal Soliton distributions are analyzed and it is explained why these distributions are not proper for degree assignment. All-At-Once distribution provides the minimization of number of symbol operations; however, it requires much more encoding symbols for decoding **[Luby, 2002], [MacKay, 2005]**. Ideal Soliton distribution theoretically minimizes both the number of symbol operations and number of used encoding symbols.

The decoding algorithm of LT codes depends on the existence of degree-one encoding symbols, which are collected in a set called the 'Ripple'. The reasonable approach to obtain a distribution assuring there is always one symbol in the Ripple is presented in **[Luby, 2002]**. However, this approach is also assessed as "unrealistic" by Luby, since it assumes that the expected size of ripple is one. Luby states that even a small variance might cause the ripple to vanish and proposes the Robust Soliton distribution **[Luby, 2002]**. The proposed distribution has two important parameters: δ and c **[Luby, 2002]**: δ is a bound on the decoding failure probability, c is a free parameter generally chosen as smaller than 1. In **[MacKay, 2005]**, the effects of these parameters are discussed briefly.

The aim of this thesis work is to focus on LT codes to observe the effects of factors like degree distribution, packet segmentation, received SNR. In addition, we consider that making analyses for point-to-point and point-to-multipoint transmission

cases will be useful to comprehend the effects of the factors encountered in different scenarios.

Chapter 2 starts with the review of the encoding and decoding algorithms of LT codes **[Luby, 2002]**. We have implemented these algorithms in MATLAB to observe the effects of degree assignment, neighbor assignment through a more practical system model by benefiting from the model presented in **[Wang, Chen, 2010]**. In Section 2.3, an analysis on All-At-Once distribution is given and Ideal Soliton approach is discussed in detail by studying **[Luby, 2002]** and **[Joshi, Rhim, Sun, Wang, 2010]**. For LT codes, degree and neighbors of each encoding symbol must be known by the decoder. In literature, there is no detailed research focusing on this issue. Only **[Luby, 2002]** and **[MacKay, 2005]** present brief explanations about introducing the degrees and neighbors to the decoder. We have summarized these explanations in Section 2.4 with our comments.

Details of the system model used in our simulations are explained in Chapter 3. In addition, some basic factors affecting the performance of LT codes are analyzed and presented as our preliminary simulation results in Chapter 3. These preliminary simulations are also used to verify the proper opeations of the simulators that we developed. In Section 3.3, we present the simulation results performed by using Ideal Soliton distribution. We observe that Ideal Soliton distribution gives unstable results as also expected by Luby **[Luby, 2002]**. In Section 3.4, we study the Robust Soliton distribution parameters in more detail and present the throughput results for different SNR values. We have performed simulations related to the packet length and the number of used packets, and then compared our results with the ones presented in **[Wang, Chen, 2010]** in Sections 3.5, 3.6 and 3.7.

In Chapter 4, some methods proposed by us or in the literature are studied to analyze LT codes more comprehensively and to increase the throughput values obtained while LT codes are used. In Sections 4.1 and 4.2, simulation results of two methods that we propose are presented: repeatedly sending degre-one encoding symbols and limiting the degrees of encoding symbols. In Section 4.3, the effect of packet erasure probability is analyzed to show the behaviour of LT codes in packet erasure channels. Change of throughput values while error correction codes are not

used is studied in Section 4.4. In Section 4.5, ‘Full Rank Decoding’ algorithm, which aims to continue decoding even if Ripple is empty, is explained.

Chapter 5 starts with the explanations of our assumptions related to a practical point-to-multipoint transmission scenario where LT codes are used. In the following sections of this chapter, simulation results related to different cases of point-to-multipoint transmission with and without LT codes are explained.

In Chapter 6, thesis conclusions are presented.

CHAPTER 2

LUBY TRANSFORM CODES

This chapter is a review on Luby Transform codes that closely follows the related literature **[Luby, 2002]**, **[MacKay, 2005]** and **[Joshi, Rhim, Sun, Wang, 2010]**. Before developing the LT Code simulator and simulating the practical scenarios, we have studied the steps of encoding and decoding algorithms and theoretical approach to LT codes. In this chapter, the summary of our literature research, which constitutes the base of our simulation development, is presented.

Firstly, encoding algorithm of LT codes is explained step by step in Section 2.1. Basic concepts and definitions used for LT encoding such as "degree of an encoding symbol", "neighbor input symbols" are explained in this section. In Section 2.2, decoding algorithm is introduced with explanations of "Ripple", "covered input symbols" and "recovered input symbols". Theoretical approach to degree distribution, which has a major effect on number of required encoding symbols to decode whole data, is analyzed in Section 2.3. In this section, it is aimed to clarify the derivations of equations presented in literature. Finally, in Section 2.4, methods which can be used to inform the receiver about degrees and neighbor input symbols of each encoding symbol is summarized with our comments.

2.1. ENCODING ALGORITHM

To transmit an N-bit data sequence using LT coding, encoding is performed as follows:

1) Data sequence is partitioned into N/l encoding symbols, where l is the decided length of one encoding symbol. Each l-bit sequence created by partitioning of N-bit data is called an 'input symbol'.
2) A degree, d, is assigned to each 'encoding symbol' according to a pre-decided probability mass function. Degree can be an integer between 1 and N/l, including 1 and N/l.

3) Neighbor ‘input symbols’ are chosen for each ‘encoding symbol’. Number of neighbors of an ‘encoding symbol’ is d; i.e., the degree of that encoding symbol. This choice is performed uniformly among all input symbols.
4) After deciding on d and neighbors of the ‘encoding symbol’, all neighbors are bitwise XOR’ed. This operation is equivalent to the addition of symbol bits in modulo two.

In conclusion, the encoding process is composed of segmentation of data, assignment of degrees and selection of neighbors followed by the XOR operation.

An exemplary diagram of the encoding algorithm is given in Figure 2.1 where the length l of an encoding symbol is 1.

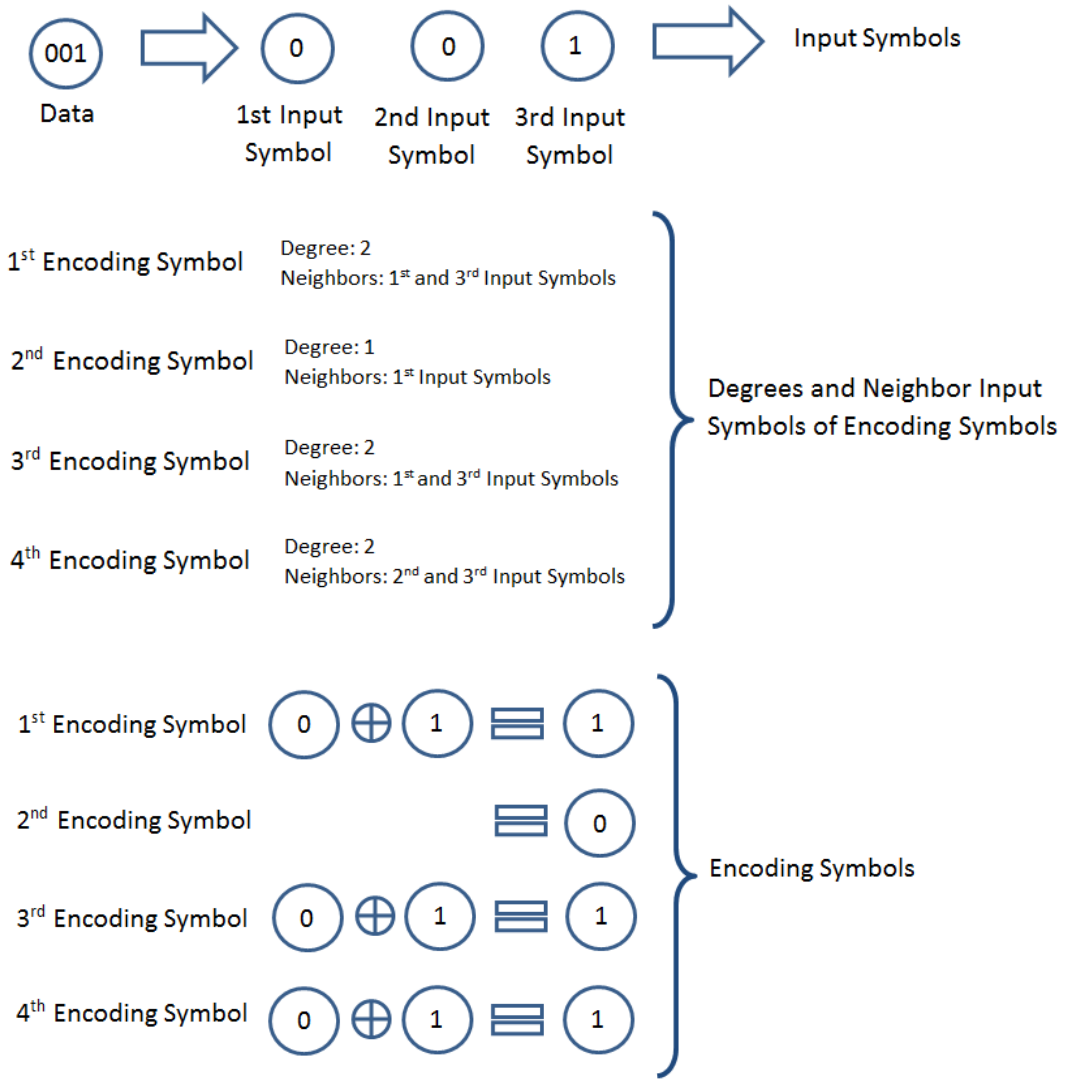


Figure 2.1: Encoding algorithm of Luby Transform codes

In the encoding algorithm, the most important part to obtain high throughput values, is the decision on the probability mass function of degree. Luby states this fact in **[Luby, 2002]** and focuses on the distribution of degrees.

In **[Luby, 2002]**, it is indicated that encoding and decoding is more efficient for larger values of l; however, no further information is given about this issue. On the other hand, effect of l is explained in more detail in **[Wang, Chen, 2010]**. In addition, the effect of number of input symbols on throughput is examined in **[Wang, Chen, 2010]**. In this thesis work, effects of symbol length and number of input symbols will also be discussed in Sections 3.5, 3.6 and 3.7.

2.2. DECODING ALGORITHM

To decode the packets encoded by using Luby Transform Coding, the following procedure is performed:

1) All encoding symbols having degree 1 are detected.
2) A set is created with each unique neighbor input symbol of each detected degree-one encoding symbol. This set is called 'Ripple'. The elements of Ripple are called the 'covered input symbols'. All covered but not processed input symbols are in Ripple at this step.
3) An input symbol is randomly chosen among the input symbols in Ripple to be processed.
4) Since a degree-one encoding symbol is a copy of a single neighbor input symbol, the chosen neighbor symbol is recovered immediately by assigning it the value of the encoding symbol.
5) The recovered input symbol is removed from the list of degree-one encoding symbols and its degree is reduced to zero.
6) Recovered input symbol is bitwise XOR'ed with all encoding symbols, which are also neighbors of that input symbol.
7) Recovered input symbol is removed as a neighbor from all of these encoding symbols. It is also removed from Ripple after it is processed.
8) After removal of the recovered input symbol, degrees of all affected encoding symbols are decreased by one.

9) After degree reduction, all encoding symbols, whose degrees are one, are released again. With the covered neighbor input symbols of these released encoding symbols, Ripple is created again. If there are more than one previously uncovered neighbor, Ripple grows.
10) Steps 1-9 are repeated until all input symbols are covered. If there remains still uncovered input symbols, it means that the decoding process fails with these received symbols. To continue the decoding process, new symbols contributing to the cover of uncovered input symbols are required.

In conclusion, the decoding algorithm is composed of releasing degree-one encoding symbols, selecting the input symbols covered (but not processed) by these encoding symbols and processing these input symbols.

An exemplary diagram of decoding algorithm is given in Figure 2.2 using the encoding symbols created in Figure 2.1.

In the diagrams presented in Figure 2.2 and previous explanations regarding the decoding algorithm, it is assumed that degrees and neighbor lists of encoding symbols are known by the receiver. There are several methods which can be used in practical applications to inform the decoder about degrees and neighbor lists. These methods are discussed in Section 2.4.

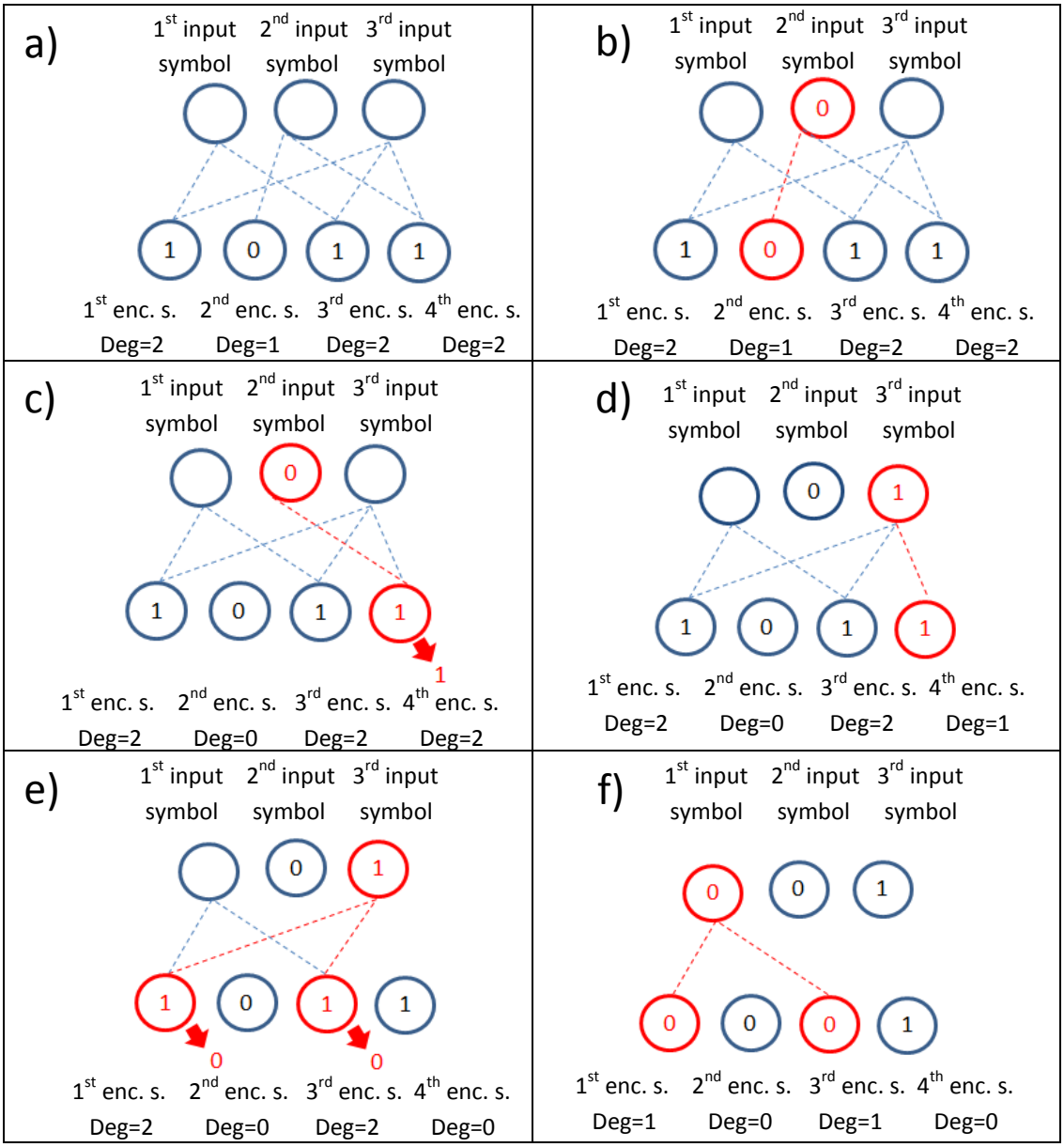


Figure 2.2: Decoding algorithm of Luby Transform codes (Reproduced from **[Tirronen, Virtamo, 2006]**)

Explanation of the steps indicated in Figure 2.2 is as follows:

a) Match the encoding symbols and input symbols according to known degrees and neighbor lists.

b) Detect a degree-one encoding symbol and assign its value to its neighbor input symbol (2^{nd} input symbol is recovered).

c) Remove the recovered input symbol as a neighbor from degree-one encoding symbol. Make the degree of this encoding symbol zero. XOR this recovered input symbol into the encoding symbols, one of whose neighbor input symbols is that input symbol.
d) Remove the recovered input symbol as a neighbor from these XOR'ed encoding symbols. Decrease the degrees of these encoding symbols by one.
Detect a degree-one encoding symbol and assign its value to its neighbor input symbol (3^{rd} input symbol is recovered).
e) Remove the recovered input symbol as a neighbor from degree-one encoding symbol. Make the degree of this encoding symbol zero. XOR the recovered input symbol into the encoding symbols, one of whose neighbor input symbols is that input symbol.
f) Remove the recovered input symbol as a neighbor from these XOR'ed encoding symbols. Decrease the degrees of these encoding symbols by one.
Detect degree-one encoding symbols and assign their value to their neighbor input symbol (1^{st} input symbol is recovered). These two encoding symbols agree on the value of this input symbol as expected.

2.3. DEGREE DISTRIBUTIONS

As previously indicated, the performance of the LT process strongly depends on the degree distribution of encoding symbols. Let the probability mass function (pmf) of degrees be $\rho(.)$. In Luby's paper **[Luby, 2002]** $\rho(d)$ is defined as the probability that an encoding symbol has degree d, where d is an integer, $1 \leq d \leq k = N/l$.

There are two main criteria to efficiently design a degree distribution $\rho(.)$:

- Minimizing the average number of used encoding symbols: Average number of used encoding symbols required for the complete recovery of the data must be minimized with the designed degree distribution.
- Minimizing the average degree of the encoding symbols: Average degree is the average number of symbol operations during the generation of an

encoding symbol. Minimizing the average degree will also minimize the number of symbol operations during encoding and decoding of the data.

The consistency of "All-at-once distribution ($p(1) = 1$)" with these criteria is analyzed as follows.

All-At-Once Distribution ($\rho(1) = 1$)

Probability mass function $\rho(1) = 1$ is also commonly named as "all-at-once distribution" because of balls and bins analogy. In balls and bins analogy, it is assumed that balls are thrown until all bins have at least one ball. Here, balls are analogous to 'encoding symbols' and bins are analogous to 'input symbols' **[Luby, 2002], [MacKay, 2005]**. It is assumed that number of balls is n and number of bins is k, which is a large number such as 1000.

When a ball is thrown, the probability of being empty for a specific bin is

$$\left(1 - \frac{1}{k}\right) \simeq e^{-\frac{1}{k}} \tag{2.1}$$

according to the power series expansion of $e^{-\frac{1}{k}}$, where $-\frac{1}{k}$ is small.

When n balls are thrown, the probability of still being empty for this specific bin is

$$\left(1 - \frac{1}{k}\right)^n \simeq \left(e^{-\frac{1}{k}}\right)^n = e^{-\frac{n}{k}} \tag{2.2}$$

If the probability that 'at least one bin is empty' is smaller than δ, which implies the probability that 'all bins have at least one ball' is at least $(1 - \delta)$, then

$$\begin{gathered}\binom{k}{1} e^{-\frac{n}{k}}\left(1 - e^{-\frac{n}{k}}\right)^{k-1} + \binom{k}{2}\left(e^{-\frac{n}{k}}\right)^2\left(1 - e^{-\frac{n}{k}}\right)^{k-2} + \cdots + \\ \binom{k}{k-1}\left(e^{-\frac{n}{k}}\right)^{k-1}\left(1 - e^{-\frac{n}{k}}\right) + \binom{k}{k}\left(e^{-\frac{n}{k}}\right)^k \leq \delta\end{gathered} \tag{2.3}$$

$$\Rightarrow k e^{-\frac{n}{k}}\left(1 - e^{-\frac{n}{k}}\right)^{k-1} < \delta \tag{2.4}$$

If $e^{-\frac{n}{k}}$ is negligibly small $\Rightarrow ke^{-\frac{n}{k}}\left(1-e^{-\frac{n}{k}}\right)^{k-1} \simeq ke^{-\frac{n}{k}}$ (2.5)

$$\Rightarrow ke^{-\frac{n}{k}} < \delta \quad (2.6)$$

$$\Rightarrow e^{-\frac{n}{k}} < \frac{\delta}{k} \quad (2.7)$$

$$\Rightarrow -\frac{n}{k} < ln\left(\frac{\delta}{k}\right) \quad (2.8)$$

$$\Rightarrow n > kln\left(\frac{k}{\delta}\right) \quad (2.9)$$

Inequality (2.9) is given in Luby's paper **[Luby, 2002]**. In addition, Equation (2.2) is included in MacKay's paper **[MacKay, 2005]**. All other steps are inserted to explain balls and bins analogy more comprehensively.

The result in (2.9) shows that the number of required encoding symbols is larger than $\ln(k/\delta)$ times the number of input symbols for all-at-once distribution. This number is undesirably high since it is intended to make the number of encoding symbols as close to the number of input symbols as possible. On the other hand, all-at-once distribution minimizes the number of symbol operations, since the sum of encoding symbol degrees indicates the total number of symbol operations. However, minimizing only the number of symbol operations is not a sufficient criterion to be proper for degree assignment **[Luby, 2002]**. Hence, all-at-once distribution cannot be assessed as an efficient distribution for Luby Transform coding.

Ideal Soliton Distribution

As indicated in the previous analysis, all-at-once distribution is not an efficient distribution in terms of number of used encoding symbols. To obtain an efficient distribution, number of used encoding symbols must be as close as possible to the number of input symbols. To ensure this, Luby states the criterion that "input symbols are added to Ripple at the same rate they are processed" (Ripple must neither grow rapidly nor disappear before the decoding process finishes) **[Luby, 2002]**. There must be only one input symbol in Ripple at each step of decoding. After the processing of this input symbol, released encoding symbol must cover an

input symbol among unprocessed input symbols to put only one input symbol into Ripple again. This also means that the expected number of released encoding symbols at each step is one **[Luby, 2002]**.

To generate a distribution making the expected number of released encoding symbols at each step one, some preliminary definitons and calculations must be introduced:

- **Definition of "Encoding Symbol Release"**

When L input symbols remain unprocessed; i.e., $(k-L)^{\text{th}}$ input symbol is processed, and an encoding symbol covers one of these remaining L unprocessed input symbols, this encoding symbol is released at this step; i.e., just after the process of $(k-L)^{\text{th}}$ input symbol.

- **Definition of "Degree Release Probability"**

Let the probability of releasing an encoding symbol whose degree is i when L input symbols remain unprocessed be $q(i,L)$.

- **Definition of "Overall Release Probability"**

Let the probability of choosing an encoding symbol to be of degree i and releasing it when L input symbols remain unprocessed be $r(i,L)$. It can be expressed as

$$r(i,L) = \rho(i)q(i,L)\ , \tag{2.10}$$

where $\rho(d)$, degree pmf, is the previously defined probability that an encoding symbol has degree d, for $1 \leq d \leq k = N/l$. Then, the overall release probability becomes $\sum_i r(i,L)$. This probability is called $r(L)$.

$$r(L) = \sum_i r(i,L) \tag{2.11}$$

- **Calculation of "$q(1,k)$"**

As indicated in the decoding algorithm of Luby Transform Codes, decoding must commence with a release of encoding symbol of degree one. Therefore, at the

beginning of decoding; i.e., when there are k unprocessed input symbols, an encoding symbol of degree one is certainly released. This means that

$$q(1,k) = 1. \tag{2.12}$$

- **Calculation of "$q(i,L)$"**

If an encoding symbol has *i* neighbors and it is released when *L* unprocessed input symbols remain;

- One neighbor input symbol is among *L* unprocessed input symbols,
- One neighbor input symbol is the one processed at the $(k-L)^{\text{th}}$ iteration,
- Other $(i-2)$ input symbols are among the first $(k-L-1)$ unprocessed input symbols.

Therefore,

$$q(i,L) = \binom{L}{1}\frac{\binom{k-L-1}{i-2}}{\binom{k}{i}} = L\frac{\frac{(k-L-1)!}{(i-2)!\,(k-L-1-i+2)!}}{\frac{k!}{i!\,(k-i)!}} \tag{2.13}$$

$$q(i,L) = L\frac{\frac{(k-L-1)(k-L-2)\dots(k-L-i+2)(k-L-i+1)!}{(i-2)!\,(k-L-i+1)!}}{\frac{k(k-1)(k-2)\dots(k-i+1)(k-i)!}{i(i-1)(i-2)!\,(k-i)!}} \tag{2.14}$$

$$q(i,L) = L\frac{i(i-1)\prod_{j=0}^{i-3}(k-(L+1)-j)}{\prod_{j=0}^{i-1}(k-j)} \tag{2.15}$$

Equation (2.15) is given in **[Luby, 2002]**. Equation (2.13) is used in **[Joshi, Rhim, Sun, Wang, 2010]** to explain the derivation of (2.15) better.

The equation given by (2.15) is valid for $i = 2,3, \dots, k$ and $L = 1,2, \dots, k - i + 1$. For all other values of i and L, $q(i, L) = 0$.

- **Calculation of "$r(L)$"**

$r(L)$ is defined as overall release probability for an encoding symbol when there are L remaining unprocessed input symbols. It is intended to finalize decoding with k encoding symbols where there are k input symbols. To obtain the probability mass function ensuring this, there must be only one encoding symbol released at each iteration as indicated before. Therefore, expected number of encoding symbols released at each step is one and it can be expressed as follows:

$$k \cdot r(L) = 1 \tag{2.16}$$

(The expected number of input symbols in Ripple is also one.)

With the previous explanations and calculations, it can be said that $r(L)$ must be uniform and its value must be $1/k$ for all $L = 1,2, \dots, k$ to design the intended distribution.

- **Obtaining the Ideal Soliton Distribution $\rho(i)$**

For $r(L) = 1/k$, let us find the values of the degree pmf $\rho(i)$ for different i values by exploiting the aforementioned definitions and equations:

If $L = k$;

$$r(k) = \sum_i r(i,k) = \sum_i \rho(i) q(i,k) = \rho(1)\underbrace{q(1,k)}_{=1} + \rho(2)\underbrace{q(2,k)}_{=0 \text{ by definition}} + \cdots$$

$$\dots + \rho(k)\underbrace{q(k,k)}_{=0 \text{ by definition}} = \frac{1}{k} \tag{2.17}$$

$$\Rightarrow \rho(1) = \frac{1}{k} \tag{2.18}$$

If $L = 1$ where $k > 1$

$$r(1) = \sum_i r(i,1) = \sum_i \rho(i)q(i,k) = \rho(1)\underbrace{q(1,1)}_{= 0 \text{ by definition}} + \rho(2)q(2,1) + \cdots$$

$$\ldots + \rho(k)q(k,1) = \frac{1}{k} \tag{2.19}$$

$$= \rho(2)\frac{2 \cdot 1}{k(k-1)} + \rho(3)\frac{3 \cdot 2 \cdot (k-2)}{k(k-1)(k-2)} + \cdots$$

$$\ldots + \rho(i)\frac{i(i-1)(k-2)(k-3)\ldots(k-i+1)}{k(k-1)(k-2)(k-3)\ldots(k-i+1)} + \cdots + \rho(k)\frac{k(k-1)}{k(k-1)} = \frac{1}{k} \tag{2.20}$$

If $\rho(i)$ is chosen to be

$$\frac{1}{i(i-1)} \tag{2.21}$$

for each i, where $2 \leq i \leq k$, then:

$$r(1) = \frac{1}{2 \cdot 1}\frac{2 \cdot 1}{k(k-1)} + \frac{1}{3 \cdot 2}\frac{3 \cdot 2}{k(k-1)} + \cdots + \frac{1}{k(k-1)}\frac{k(k-1)}{k(k-1)} \tag{2.22}$$

$$= (k-1)\frac{1}{k(k-1)} = \frac{1}{k} \tag{2.23}$$

Equation (2.23) holds for the chosen distribution given in (2.21).

If (2.16) is checked with the distribution $\rho(i)$ given in (2.21) for another value of *L*:

If $L = 2$ where $k > 2$

$$r(2) = \sum_i r(i,2) = \sum_i \rho(i)q(i,k) = \rho(1)\underbrace{q(1,2)}_{= 0 \text{ by definition}} + \rho(2)q(2,2) + \cdots + \rho(k)q(k,2) = \frac{1}{k} \tag{2.24}$$

$$= \rho(2) \cdot 2\frac{2 \cdot 1}{k(k-1)} + \rho(3) \cdot 2\frac{3 \cdot 2 \cdot (k-3)}{k(k-1)(k-2)} + \rho(4) \cdot 2\frac{4 \cdot 3 \cdot (k-3)(k-4)}{k(k-1)(k-2)(k-3)} +$$

$$\ldots + \rho(k-1) \cdot 2\frac{(k-1)(k-2)(k-3)(k-4) \ldots 2 \cdot 1}{k(k-1)(k-2) \ldots 3 \cdot 2} + \rho(k) \cdot 0 = \frac{1}{k}$$

(2.25)

If $\rho(i)$is chosen to be $\frac{1}{i(i-1)}$ again for each i, where $2 \leq i \leq k$

$$r(2) = \frac{2}{k(k-1)} + \frac{2(k-3)}{k(k-1)(k-2)} + \frac{2(k-4)}{k(k-1)(k-2)} + \cdots + \frac{2 \cdot 1}{k(k-1)(k-2)}$$

(2.26)

$$= \frac{2}{k(k-1)} + \frac{2}{k(k-1)(k-2)}[(k-3) + (k-4) + (k-5) + \cdots + 1]$$

(2.27)

$$= \frac{2}{k(k-1)} + \frac{2}{k(k-1)(k-2)} \sum_{j=1}^{k-3} j \quad (2.28)$$

$$= \frac{2}{k(k-1)} + \frac{2}{k(k-1)(k-2)} \frac{(k-3)(k-2)}{2} \quad (2.29)$$

$$= \frac{2}{k(k-1)} + \frac{k-3}{k(k-1)} = \frac{k-1}{k(k-1)} = \frac{1}{k} \quad (2.30)$$

Equation (2.16) holds again as it is expected.

If it is checked whether $r(L) = \frac{1}{k}$ for other values of L with $\rho(i)$ given in (2.31), it can be seen that (2.16) holds for all L values.

$$\rho(i) = \begin{cases} \frac{1}{k} & if\ i = 1 \\ \frac{1}{i(i-1)} & if\ i = 2,3,\dots,k \end{cases} \tag{2.31}$$

Equations from (2.17) to (2.30) are presented to prove that (2.31) theoretically satisfies the criterion that expected size of the ripple at each step is 1.

The probability mass function given by (2.31), which also ensures $\sum_i \rho(i) = 1$ as any pmf does, is the one required for efficiency. Theoretically, k input symbols can be covered with k encoding symbols if Ideal Soliton Distribution is used, unlike the case where All-At-Once distribution is used. However, this distribution is not sutiable for practical applications because even a small variation causes Ripple to vanish **[Luby, 2002]**. Therefore, "one input symbol in Ripple at each iteration" requirement is not met. In this case, decoding fails.

Robust Soliton Distribution $\mu(i)$

In Luby's paper, Robust Soliton pmf $\mu(i)$ is defined as follows **[Luby, 2002]**:

$$\tau(i) = \begin{cases} \frac{R}{ik}\ for\ i = 1,2,\dots,\frac{k}{R} - 1 \\ \frac{R}{k} ln\left(\frac{R}{\delta}\right) \qquad for\ i = \frac{k}{R} \\ 0\ for \quad i = \frac{k}{R} + 1,\dots,k \end{cases} \tag{2.32}$$

$$\beta = \sum_{i=1}^{k} [\rho(i) + \tau(i)] \tag{2.33}$$

$$\mu(i) = \frac{\rho(i) + \tau(i)}{\beta} \tag{2.34}$$

where $\rho(i)$ is the Ideal Soliton Distribution given by (2.31), $R = c \cdot \ln\left(\frac{k}{\delta}\right)\sqrt{k}$ for some constant $c > 0$. δ is defined as 'allowable failure probability of the decoder' which is also used in balls and bins analogy to explain All-At-Once Distribution. The effects of parameters of Robust Soliton Distribution will be discussed in the following chapters.

As previously indicated, All-At-Once Distribution and Ideal Soliton Distribution are not suitable for practical applications since the required number of used encoding symbols is higher than the possible minimum value for All-At-Once Distribution and even a small variation in the size of Ripple might cause the Ripple to vanish before decoding ends for Ideal Soliton Distribution. On the other hand for Robust Soliton Distribution, the size of the Ripple remains large enough at each point of the decoding process to ensure that Ripple does not vanish before decoding ends with high probability. In addition, the Robust Soliton Distribution also helps to minimize the size of the Ripple and the number of the used encoding symbols by preventing the release of too many encoding symbols redundantly covering the input symbols which already exist in the Ripple. The idea used for designing the Robust Soliton Distribution is to keep the Ripple size about $\ln\left(\frac{k}{\delta}\right)\sqrt{k}$ during the decoding process **[Luby, 2002]** for k input symbols and allowable decoding failure probability δ. Comparisons between Ideal Soliton Distribution and Robust Soliton Distribution will also be presented in the following chapters.

2.4. INTRODUCING DEGREE AND NEIGHBOR LISTS TO THE DECODER

To continue the decoding process properly, degree and neighbors of each encoding symbol must be exactly known by the decoder. In the literature, especially in Luby's and MacKay's papers, several ways are presented to inform the decoder about degrees and neighbors of the encoding symbols **[Luby, 2002], [MacKay, 2005]**, to be mentioned briefly in this section.

Degree and neighbor information can be explicitly transmitted to the decoder through each packet. Assume each packet has a structure composed of a header and actual data aimed to be transmitted, like IP packets. Order, degree and neighbors of each encoding packet can be coded with known conventional methods other than Luby Transform. Actual data sequence placed right after the header can be coded according to Luby Transform encoding technique with the parameters explicitly indicated in the header. This method could work; however, it can be assessed as an inefficient technique for practical applications. Let us discuss why this technique is inefficient with an example: Assume that there are 1000 input packets (symbols) and

each input packet (symbol) has a length of 100 bits. Also assume that one of the encoding packets aimed to be sent has a high degree such as 20. To encode the index of each neighbor, at least 10 bits must be used (2^{10}=1024). In this case, 200 additive bits must be used to introduce the indices of neighbors of a 100-bit packet. If any other error detecting or correcting technique is also used, the number of redundant bits extremely increases.

Synchronization can also be used to introduce the degrees and neighbors. Assume that both encoder and decoder have a fixed list of degrees and neighbors; therefore, it is not required to send this information in a header explicitly. If encoder and decoder synchronize to each other successfully at the beginning of the transmission, decoder knows the required information to perform the decoding process.

Another method to inform the decoder is to use pseudo random number generators in both encoder and decoder. Pseudo random number generators generate the same results for the same inputs called ‘seeds’. If a seed is sent by the encoder through each encoding symbol, the decoder can generate degree and neighbor lists according to this seed. Therefore, synchronization is not necessarily required for this method. If any packet is erased at the beginning or during the transmission, decoder can obtain the necessary information from the seeds of non-erased packets. Degrees and indices of neighbors can be obtained by the decoder by using seeds for pseudo random number generators.

CHAPTER 3

SYSTEM MODEL AND PRELIMINARY SIMULATIONS

While developing our LT code simulator, we have firstly generated LT encoder and decoder separately. After verifying that LT encoder and decoder operate properly by checking degree distributions and decoded symbols, we have decided to use the system model used in **[Wang, Chen, 2010]** since it is a realistic model including noise and fading effects, error detecting and correcting codes. By performing some preliminary simulations with this system model using point-to-point transmission, we have compared our results with the ones presented in **[Wang, Chen, 2010]**, to validate proper operation of our simulator including the components related to practical data transmission scenarios.

Section 3.1 includes the explanation of the components constituting our system model. Calculation of throughput is explained in Section 3.2. In the subsequent sections, simulation results for different degree distributions, packet lengths and number of packets are presented, respectively.

3.1. SYSTEM MODEL

System model used in **[Wang, Chen, 2010]** that we have simulated is given in Figure 3.1.

Data is firstly segmented into packets. According to the decided length, segmentation is performed and packets are prepared. Then, the LT encoder takes segmented input packets and assigns degrees to encoding symbols according to the decided probability mass function. Neighbor input symbols are chosen and each encoding symbol is created with XOR operation of its neighbor input symbols.

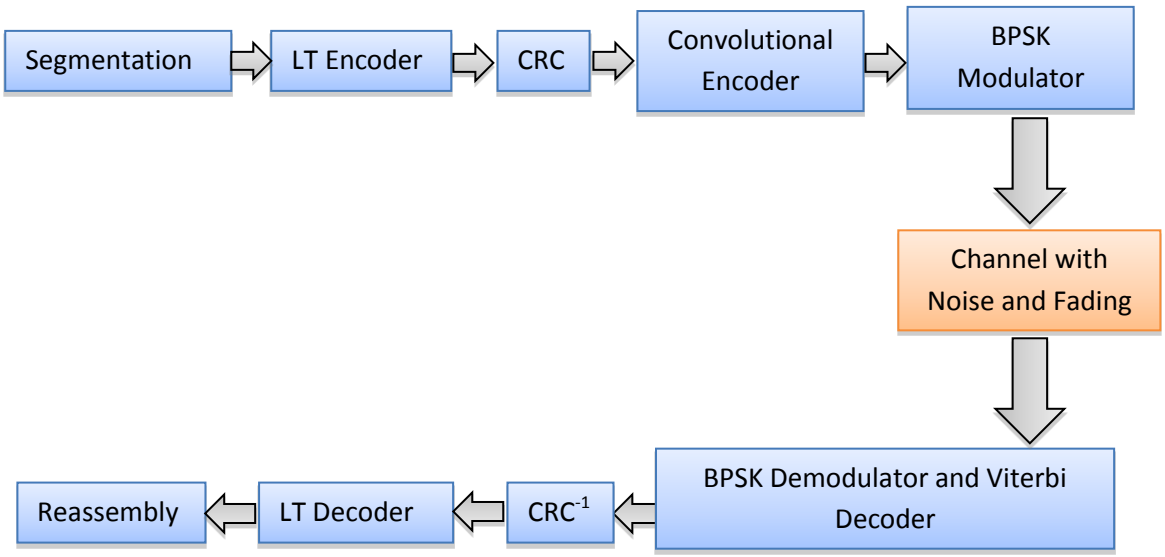


Figure 3.1: System model used for Luby Transform encoding and decoding simulations (Reproduced from **[Wang, Chen, 2010]**)

Cyclic Redundancy Check (CRC) adds $n-k$ check bits to k information bits using the generator polynomial $g(X) = 1 + X^5 + X^{12} + X^{16}$ recommended by CRC-16-CCITT, which is one of the most commonly used polynomials in practical applications (see Appendix A). The convolutional code is of rate ½ and constraint length 7, with generator polynomials $(171, 133)_8$ (Appendix A). Modulation method is chosen as BPSK.

Additive White Gaussian Noise (AWGN) is used for both in-phase and quadrature components in the system model. 'Rayleighchan' function of MATLAB is used to obtain channel coefficients. Channel coefficients are assumed to be constant for each packet; however, independent from the channel coefficients of other packets. Therefore, the time diversity is high in our simulations. This scenario is similar to a frequency hopping system and the following simulations are performed according to this scenario. Maximum Doppler frequency (f_D) is taken as 10 Hz and symbol period (T_S) is taken as 0.000001 (see Appendix B) in our simulations.

Soft decision Viterbi decoder is implemented in MATLAB, where Euclidean distance is used unlike hard decision decoding that exploits Hamming distances.

After all LT encoding packets are decoded (i.e. all input packets are found), these packets are reassambled and whole data is obtained.

Simulations based on the previously mentioned system model are performed for point-to-point transmission firstly. Although it is not reasonable to use LT codes for point-to-point transmission, as a preliminary work, our aim is to optimize the parameters which shall be used in point-to-multipoint transmission scenario. Analyses of some cases for point-to-multi point transmission are made in Chapter 5.

In the preliminary simulations, encoder is assumed to be able to send packets continuously until decoder decodes all the data for point-to-point transmission. In practical applications, encoder must stop transmission somehow to start sending a new data sequence. It can be done through a feedback packet sent by decoder when all packets are decoded. Therefore, it is assumed that the decoder sends a feedback when it decodes whole data for point-to-point transmission scenarios. It can be evaluated as a simple ARQ mechanism, where an acknowledgment is sent from the receiver to the transmitter only when all packets are decoded successfully. In this scenario, when the packets which do not utilize LT codes are used, these packets are transmitted in order. After the transmitter sends all packets, it waits for an acknowledgment. If it does not receive any acknowledgment, it transmits all packets in order again. We call this case 'point-to-point with no LT code'. When the packets utilizing LT codes are used, transmitter waits for an acknowledgment indicating that the whole data is decoded again; however, packets are sent as random functions of input symbols instead of being sent in order. We call this case 'point-to-point with LT code'. Point-to-point system is defined according to the aforementioned explanations since we aimed to verify the proper operation of our simulator by comparing the results of our preliminary simulations with the results presented in **[Wang, Chen, 2010]**, which uses the same system model.

In broadcasting applications, sending feedback is not meaningful. A method which can be used to stop the encoder is to limit the number of encoding symbols to be sent to a maximum value. In this case, the decoder might not be able to decode all the packets. Let us call this scenario, where the transmitter stops sending packets after sending a certain number of packets 'point-to-multipoint'. When the packets which do not utilize LT codes are used, these packets are transmitted in order until

the number of sent packets reaches to the defined maximum value. We call this case 'point-to-multipoint with no LT code'. When the packets utilizing LT codes are used, the transmitter sends the packets until the number of sent packets reaches to the defined maximum value; however, packets are sent as random functions of input symbols instead of being sent in order. We call this case 'point-to-multipoint with LT code'. Point-to-multipoint system is defined according to the aforementioned explanations since we aimed to construct a more practical system model, which can be encountered in real life broadcasting scenarios. Analyses of some cases related to point-to-multipoint transmission are also made for limited number of encoding symbols and probabilities of decoding for defined limited numbers. These results are presented in Chapter 5.

3.2. THROUGHPUT CALCULATION

For all the simulations performed according to the aforementioned system model, throughput calculation is made as follows:

$$\eta = \frac{k}{k'} \cdot \frac{l}{l'} \cdot \frac{l'}{l''} \cdot \frac{k_c}{n_c}$$

$\eta: Throughput,$
$k: Number\ of\ inputs\ symbols,$
$k': Total\ number\ of\ transmitted\ encoding\ symbols,$
$l: Length\ of an\ input\ symbol,$
$l': Length\ of\ a\ symbol\ with\ CRC\ encoding\ (l' = l + 16),$
$l'': Length\ of\ a\ padded\ symbol\ for\ convolutional\ encoding\ (l'' = l' + 6),$
$\frac{k_c}{n_c}: Coding\ rate\ of\ convolutional\ encoding\ (\frac{k_c}{n_c} = \frac{1}{2})$

3.3. IDEAL SOLITON DISTRIBUTION

As mentioned before, Ideal Soliton Distribution is not a proper distribution for practical applications since it might cause the set called 'Ripple' either to vanish or to grow too large before decoding ends where a certain number of encoding symbols are used. In both cases, "one input symbol in Ripple at each decoding step" principle does not hold; therefore, decoding fails.

In Figure 3.2(a), closed form pmf expression given by (2.31) of Ideal Soliton Distribution is plotted. Figure 3.2(b) shows the actual distribution of encoding symbol degrees for one of the trials performed with Ideal Soliton Distribution. It also shows that the degree assignment process is performed properly with the MATLAB code written for this simulation.

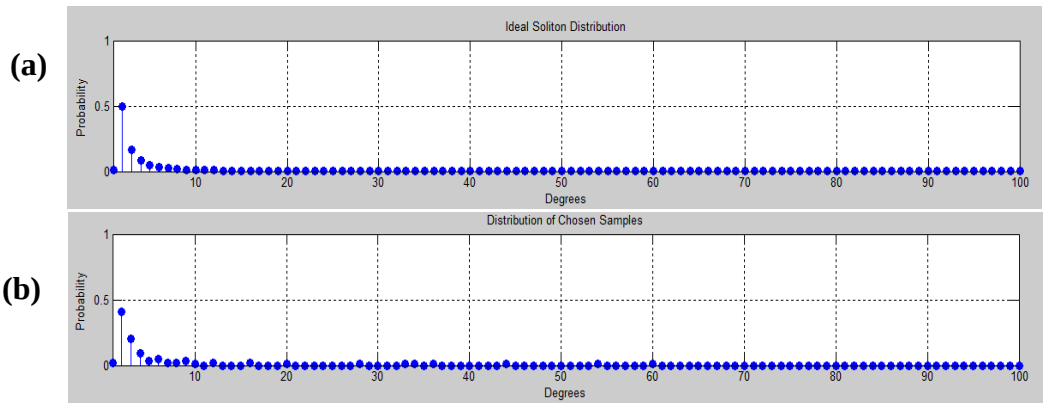


Figure 3.2: **a)** Plot of closed form expression of Ideal Soliton Distribution **b)** A sample degree distribution for a trial over 100 symbols

To assign a degree to each encoding symbol, "inverse cumulative mass function" of Ideal Soliton Distribution is used. Let $f(x)$ be any probability mass function and $F(x)$ be the cumulative mass function corresponding to this probability mass function, where x is the random variable defined for encoding symbol degree, whose minimum value is 1 and maximum value is the number of input symbols. Then, as shown in Figure 3.3, if any value a is chosen between 0 and 1 uniformly, where $a_1 < a \leq a_2$, $F(x-1) = a_1$ and $F(x) = a_2$, the degree of the encoding symbol for which uniform number generator generates the number a is selected as x.

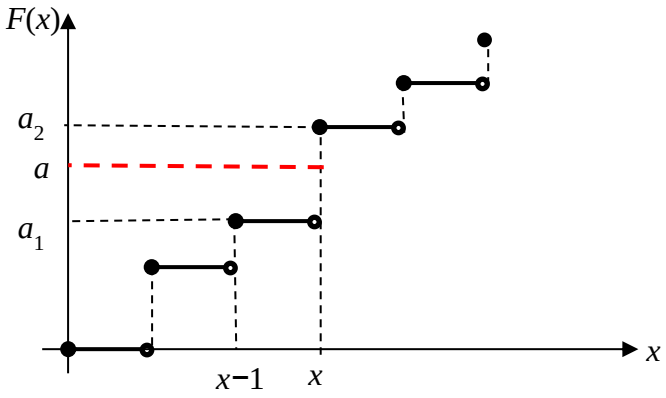


Figure 3.3: Exemplary cumulative mass function for assignment of degrees of encoding symbols

Figure 3.4 shows an exemplary degree assignment for Ideal Soliton Distribution where there are 100 input symbols and 300 encoding symbols prepared to be sent. This figure shows that when there are 100 input symbols there can be several encoding symbols whose degrees are high, such as 40 or more. This made us consider that these encoding symbols are useless and performance of LT codes might increase if these symbols are not used; i.e., if degrees of encoding symbols are limited. Effects of 'degree limitaton' is discussed in Chapter 4.

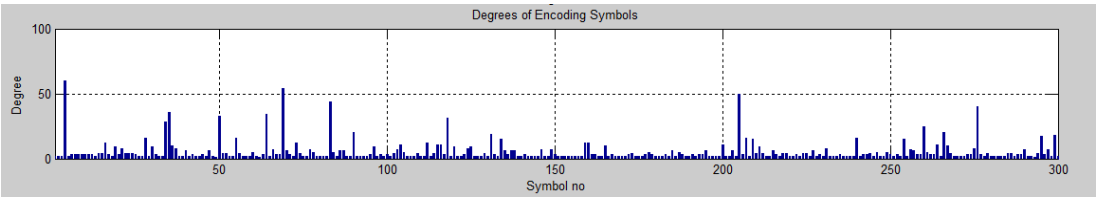


Figure 3.4: Exemplary degree assignment of 300 encoding symbols according to Ideal Soliton Distribution for 100 input symbols

In Figure 3.5, throughput values for different SNR values of received signals are given, where there are 100 input symbols each consisting of 100 bits and 100 trials are performed for each SNR value. As it can be seen from Figure 3.5, for high SNR's throughput values change between 0.2 and 0.26 during simulations. This change is due to the nature of Ideal Soliton Distribution. In some cases, more encoding symbols are used since Ripple contains no input symbols. This excessively

decreases the average value of obtained throughput at some SNR values randomly. For low SNR values, decoding cannot be completed.

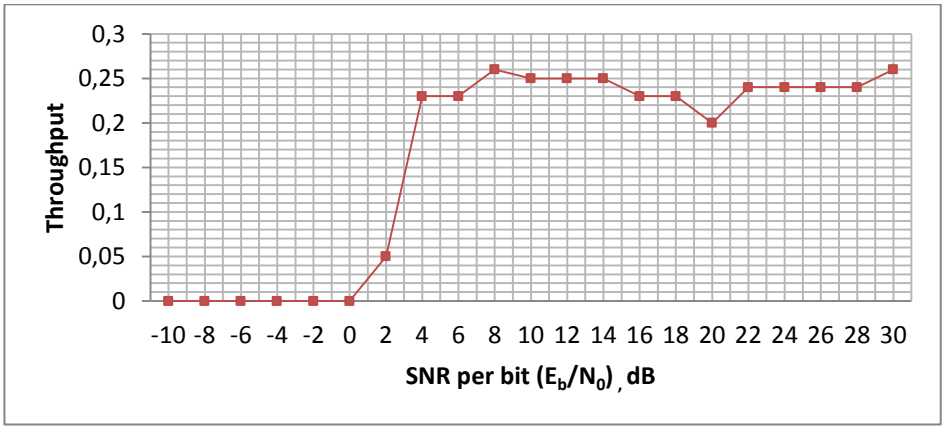


Figure 3.5: Throughput values vs SNR for 'point-to-point with LT code' scenario and Ideal Soliton Distribution

3.4. ROBUST SOLITON DISTRIBUTION PARAMETERS

As mentioned before, Robust Soliton Distribution, $\mu(i)$, is composed of the Ideal Soliton Distribution $\rho(i)$, $\tau(i)$ and β, whose definitions are given in (2.32)-(2.34) as:

$$\tau(i) = \begin{cases} \frac{R}{ik} \; for \; i = 1,2,\dots,\frac{k}{R} - 1 \\ \frac{R}{k} ln\left(\frac{R}{\delta}\right) \qquad for \; i = \frac{k}{R} \\ 0 \; for \quad i = \frac{k}{R} + 1,\dots,k \end{cases}$$

$$\beta = \sum_{i=1}^{k} [\rho(i) + \tau(i)]$$

$$\mu(i) = \frac{\rho(i)+\tau(i)}{\beta}$$

where $R = c \cdot \ln\left(\frac{k}{\delta}\right)\sqrt{k}$ is the expected Ripple size for some constant $c > 0$. δ is the allowable failure probability of the decoder. Definition of $\tau(i)$ at point $i = \frac{k}{R}$ causes the second spike in probability mass function plot. Therefore, Ripple neither vanishes

nor grows too large before decoding ends. Because of this second spike, Robust Soliton Distribution gives more stable and efficient results for practical applications.

For Robust Soliton Distribution, it is important to choose the values of c and δ properly. In **[MacKay, 2005]**, there is an analysis on number of degree-one encoding packets and total number of required encoding packets for different c and δ values regardless of received SNR; i.e., it is assumed that all packets are received correctly. In Figure 3.6(a), change of generated degree-one encoding symbols according to changing values of c and δ is given for k=10000 input symbols. Figure 3.6(b) presents the change of the total number of required encoding packets for k=10000 input symbols. Depending on the results presented in Figure 3.6(b), it can be generally said that throughput increases when δ (delta) increases and c decreases, since number of required encoding packets decreases.

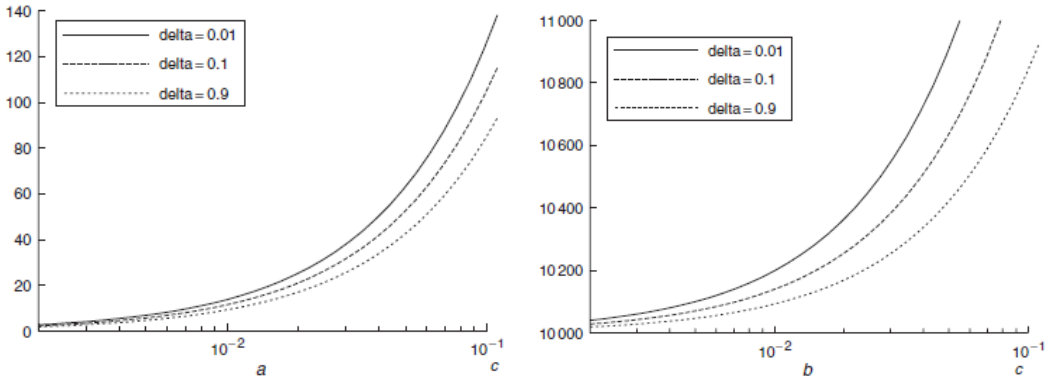


Figure 3.6: a) Number of degree-one encoding symbols **b)** Total number of required encoding packets for k=10000 input symbols (Reproduced from **[MacKay, 2005]**)

In our preliminary simulations, effects of c and δ are observed over 20 trials in addition to the locations of the first and second spikes in Robust Soliton Distribution, the expected degrees and the average number of required encoding symbols; for k=100 and 1000 input symbols. Values used in our simulations are given in Table 3.1.

Table 3.1: c and δ values used in our simulations and corresponding values of locations of spikes, expected degree and average number of required encoding symbols over 20 trials

k	c	δ	Location of 1st spike	Location of 2nd spike	Expected degree	Average number of required packets
100	0.01	0.05	2	132*	5.72	141
100	0.01	0.5	2	189*	5.56	145
100	0.1	0.05	2	13	6.84	147
100	0.1	0.5	2	19	6.5	133
100	0.5	0.05	2	3	3.3	153
100	0.5	0.5	2	4	4.02	135
1000	0.01	0.05	2	319	12.21	1166
1000	0.01	0.5	2	416	9.86	1150
1000	0.1	0.05	2	32	11.23	1259
1000	0.1	0.5	2	42	6.9	1205
1000	0.5	0.05	2	6	6.05	1632
1000	0.5	0.5	2	8	6.9	1507

**: Second spike cannot be observed in Robust Soliton Distribution plot since this number is higher than the number of input symbols*

By analyzing the presented results, the observation obtained from **[MacKay, 2005]** can be detailed as follows: throughput increases when c decreases and δ increases, as long as the location of the second spike is smaller than the number of input symbols; i.e. the second spike can be observed. Otherwise, characteristics of the obtained distribution is similar with the characteristics of the Ideal Soliton Distribution. For instance, if $k = 100$, $c = 0.01$ and $\delta = 0.5$, $\mu(i)$ should have two spikes at $i = 2$ (because of $\rho(i)$) and $i = \frac{100}{0.01 \cdot \ln\left(\frac{100}{0.5}\right)\sqrt{100}} \cong 189$ (because of $\tau(i)$). However, the second spike cannot be observed in Robust Soliton Distribution plot since the maximum degree can be 100 when 100 input symbols are used. In Figure 3.7(a) and (b), closed form expression plot and actual distribution of degrees are given. As it can be seen from this figure, distribution is similar to the Ideal Soliton Distribution.

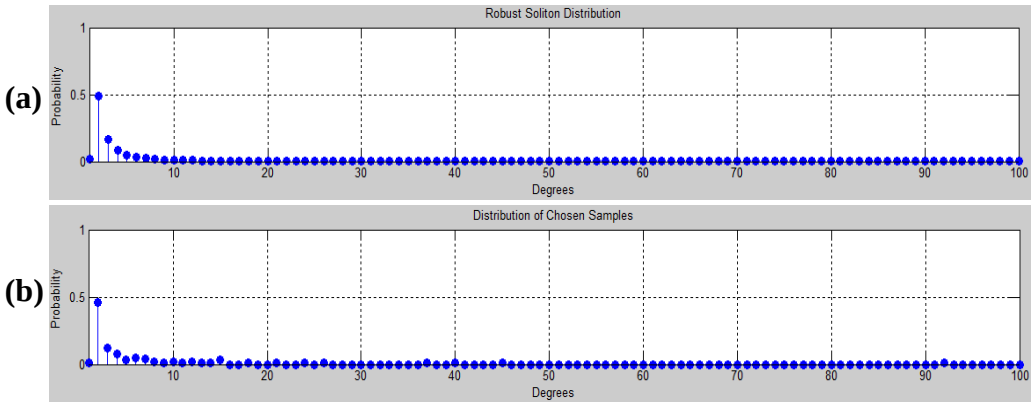


Figure 3.7: a) Plot of closed form expression of Robust Soliton Distribution for $k = 100$, $c = 0.01$ and $\delta = 0.5$ **b)** A sample distribution of encoding symbol degrees for a trial over 100 symbols

However, for $k = 100$, $c = 0.1$ and $\delta = 0.5$, $\mu(i)$ has two spikes at $i = 2$ (because of $\rho(i)$) and $i = \frac{100}{0.1 \cdot \ln\left(\frac{100}{0.5}\right)\sqrt{100}} \cong 19$ (because of $\tau(i)$). In Figure 3.8(a) and (b), plots of closed form expression of Robust Soliton Distribution and actual distribution of encoding symbol degrees for these values are given, respectively. The second spike can be observed in these plots.

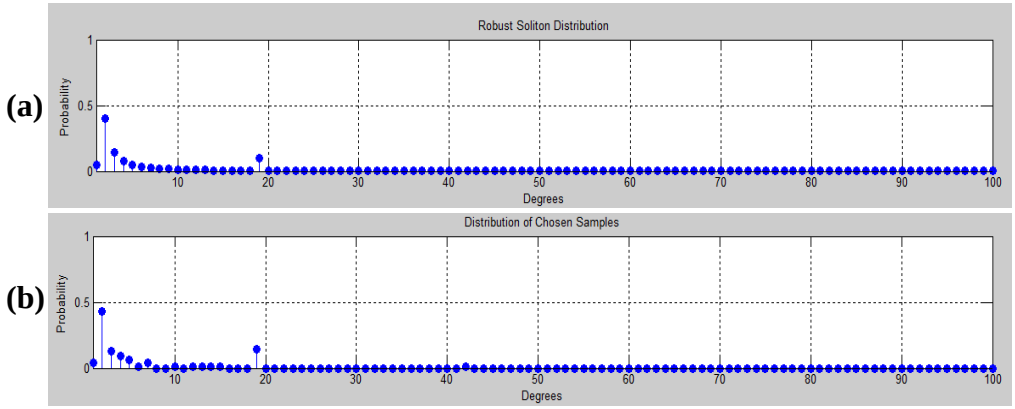


Figure 3.8: a) Plot of closed form expression of Robust Soliton Distribution for $k = 100$, $c = 0.1$ and $\delta = 0.5$ **b)** A sample distribution of encoding symbol degrees for a trial over 100 symbols

With all these observations it can be said that the number of required encoding symbols, which is inversely related to the throughput, does not only depend

on the increase or decrease of c and δ values, but also on the number of input symbols and the location of the second spike. However, it cannot be said that an increase or decrease of the expected degree directly affects the number of required encoding symbols to decode the whole data.

In Figure 3.9(a) and (b), average numbers of required encoding symbols with varying values of c and δ for 100 and 1000 input symbols are sketched over 20 trials, respectively.

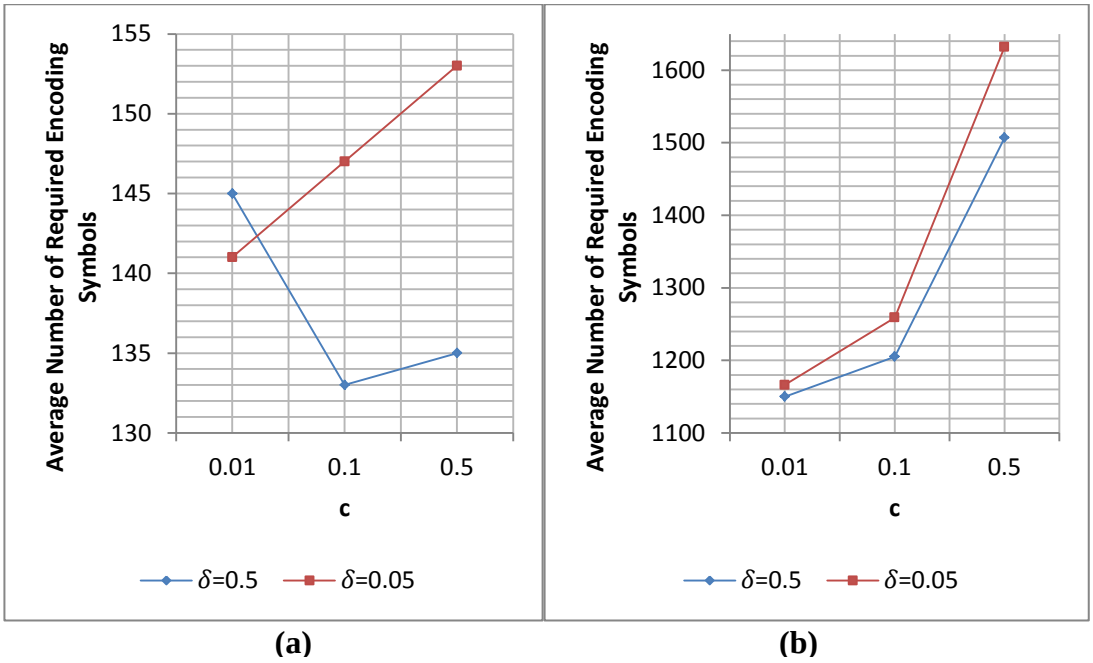
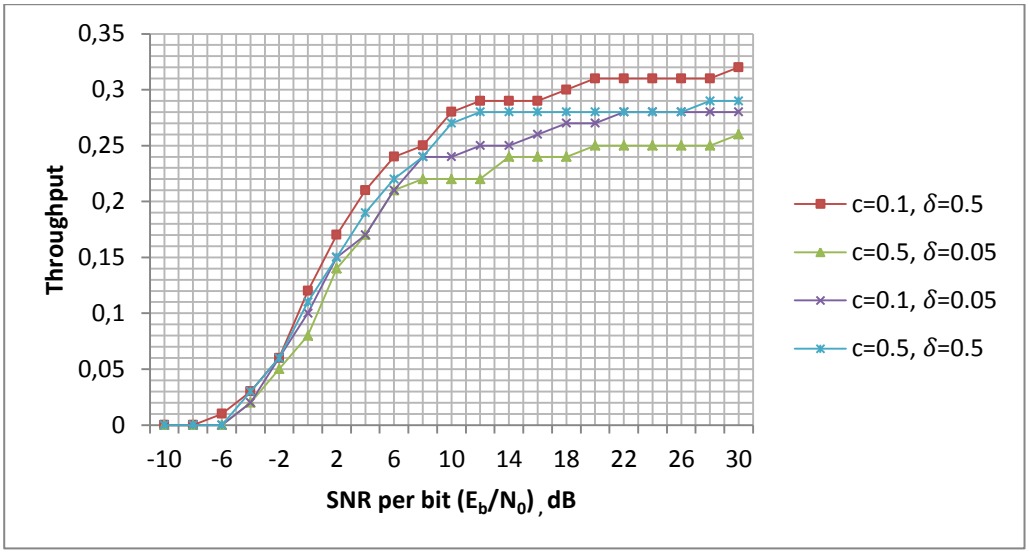


Figure 3.9: Average numbers of required encoding symbols for **a)** k=100 **b)** k=1000

Simulations performed over 20 trials to compare these different c and δ values in terms of the obtained throughput for k=100 are given in Figure 3.10(a) and (b). In Figure 3.10(a), four combinations of c and δ giving more unwavering throughputs are presented. In Figure 3.10(b), the comparison between two combinations of c and δ giving fluctuating results like Ideal Soliton Distribution and the (c=0.1, δ=0.5) combination giving the steadiest thoughputs for k=100 is made.

(a)



(b)

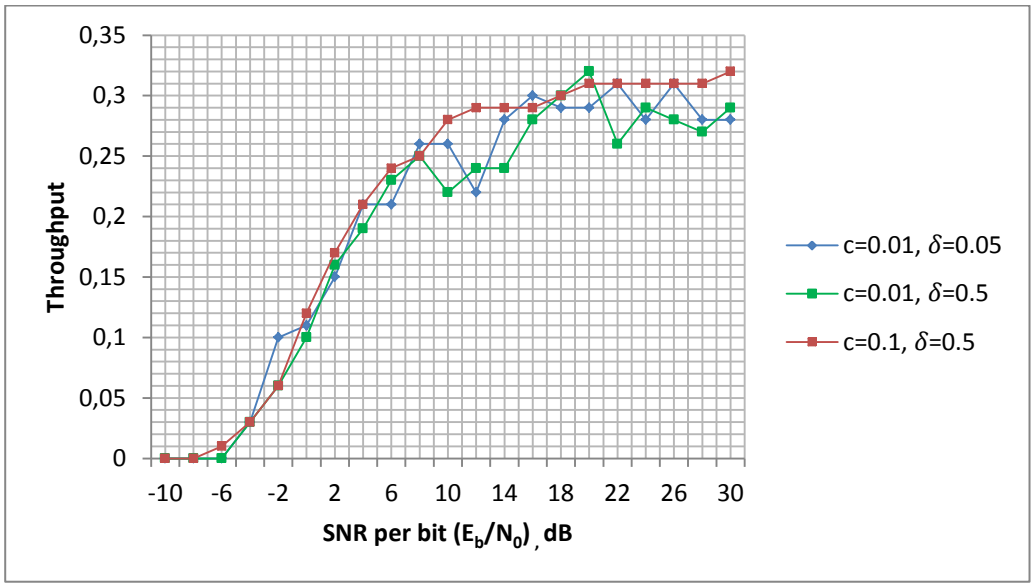


Figure 3.10: For 'point-to-point with LT code' scenario and Robust Soliton Distribution, **a)** Comparison of throughput values for c and δ values giving more steady results, **b)** Comparison of throughput values for (c=0.1, δ=0.5), (c=0.01, δ=0.5) and (c=0.01, δ=0.05); for number of input symbols k=100.

As it can be seen from 'part a' of Figure 3.10, if δ decreases throughput also decreases; i.e., the number of required encoding packets to successfully decode the data increases as long as the second spike can be observed. On the other hand, if c decreases throughput increases; i.e., the number of required encoding packets to successfully decode the data decreases. For the combinations of δ and c that we analyzed, the best combination is (c=0.1, δ=0.5) and the worst combination is (c=0.5, δ=0.05) in terms of throughput when k=100. For some SNR values,

throughput value for the best combination is about 20% higher than the throughput value for the worst combination. Another observation related to Figure 3.10(a) is that Robust Soliton Distribution with proper values of parameters gives a much more stable throughput change with varying SNR values; i.e. throughput increases with increasing SNR, unlike Ideal Soliton Distribution.

We have also found the minimum, maximum and average numbers of required encoding symbols for different Robust Soliton Distribution parameters by ignoring the channel effects; i.e., assuming all packets are received correctly. Results are presented in Table 3.2.

Table 3.2: Minimum, maximum and average numbers of required encoding symbols for different c and δ values (k=100 and 1000, l=100)

	Minimum for k=100	Maximum for k=100	Average for k=100	Minimum for k=1000	Maximum for k=1000	Average for k=1000
c=0.01, δ=0.05	112	219	141	1121	1298	1166
c=0.01, δ=0.5	109	212	145	1107	1283	1150
c=0.1, δ=0.05	116	174	147	1203	1513	1259
c=0.1, δ=0.5	115	156	133	1188	1479	1205
c=0.5, δ=0.05	126	215	153	1587	1671	1632
c=0.5, δ=0.5	111	186	135	1480	1553	1507

In Figure 3.11, a comparison between throughput values of Ideal Soliton Distribution and Robust Soliton Distribution is presented for k=100 and l=100. As it can be seen from this figure, the value of minimum required SNR to decode the whole data is higher for the case where Ideal Soliton Distribution is used. Furthermore, throughput characteristics of the case where Robust Soliton Distribution is used are more stable when compared to Ideal Soliton Distribution.

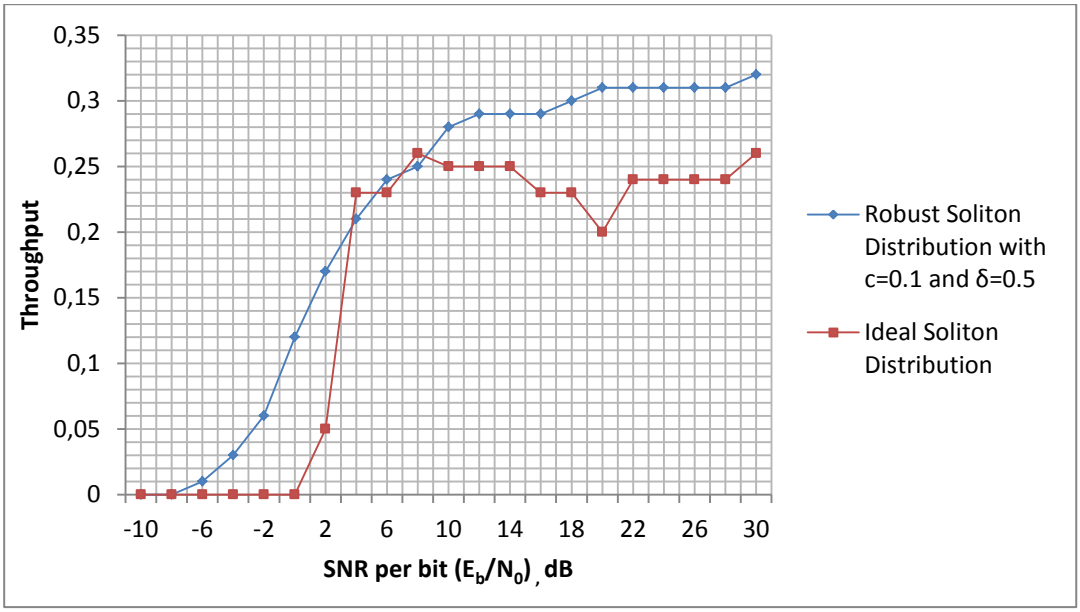


Figure 3.11: A comparison between throughput values of Ideal Soliton Distribution and Robust Soliton Distribution where *k*=100 and *l*=100 for 'point-to-point with LT code' scenario.

3.5. SIMULATION RESULTS FOR DIFFERENT PACKET LENGTHS

As indicated previously, Luby expects more efficient decoding process in terms of throughput for larger values of the packet length *l* **[Luby, 2002]**. Wang and Chen analyze the effect of packet length in detail **[Wang, Chen, 2010]**. In this thesis, we perform similar simulations and obtain similar results.

Figure 3.12 presents our simulated throughput values over 20 trials at different SNR values and packet lengths for Robust Soliton Distribution. As indicated in **[Luby, 2002]**, packets with higher lengths cause higher throughput values. However, as discussed in **[Wang, Chen, 2010]**, longer packets are more prone to be erased. For lower SNR values, shorter packet lengths create higher throughputs as shown in Figure 3.12.

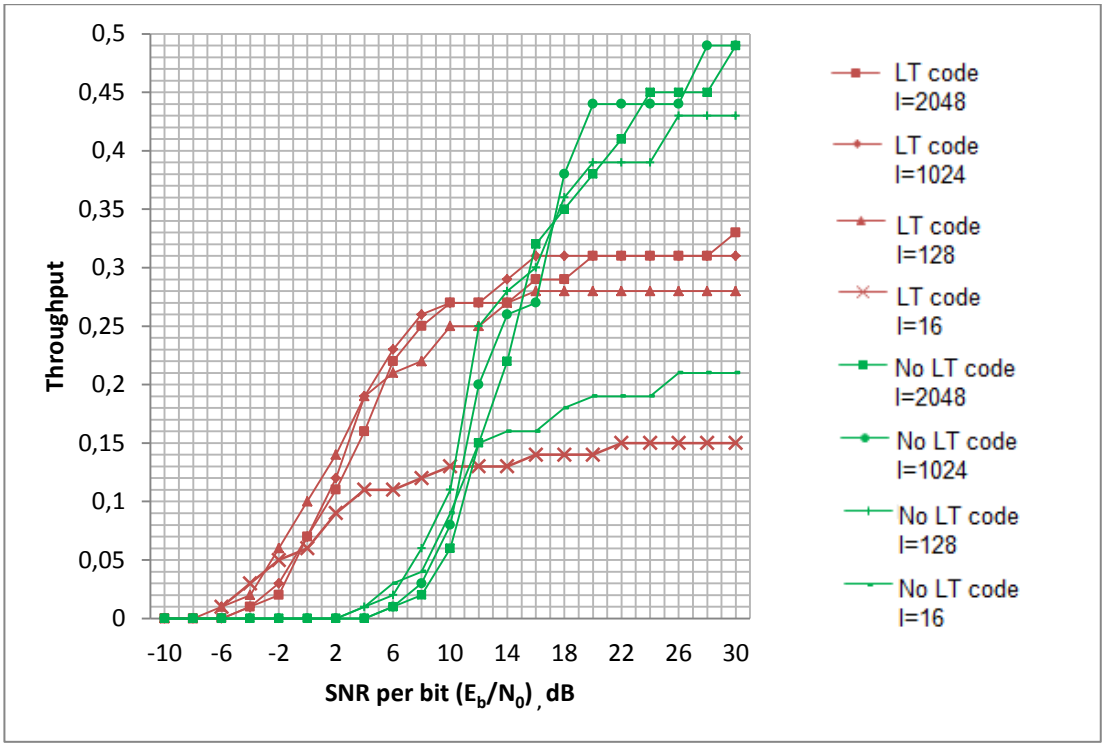


Figure 3.12: Throughput values vs SNR for point-to-point scenario with 'no LT', and 'LT with Robust Soliton Distribution' (k=10, $c = 0.1$ and $\delta = 0.5$) for different packet lengths, l.

In **[Wang, Chen, 2010]**, the performance of LT codes is compared to the case with 'no LT codes', where all packets are sent in order without LT coding. If a single packet cannot be decoded by the decoder, all packets are retransmitted again in order. This is the case that we define as 'point-to-point with no LT code' in Section 3.1. Figure 3.12 also shows the throughput results for this case. It can be said that point-to-point scenario with no LT codes is more efficient for high SNR values; however, for lower SNR values Luby Transform encoding is much more efficient. This observation can be used in some applications, where battery life of the transmitter is important. Transmission power can be intentionally decreased in these applications since in some received SNR range, encoding packets can still be decoded if LT codes are used.

3.6. SIMULATION RESULTS FOR DIFFERENT PACKET NUMBERS

In **[Wang, Chen, 2010]** it is indicated that if *k*, the number of input packets increases, throughput also increases. The results of our preliminary simulations performed with *l* =100-bit packets for different *k* values also support this observation. In Figure 3.13, our simulation results averaged over 20 trials for point-to-point scenario with and without LT codes are given.

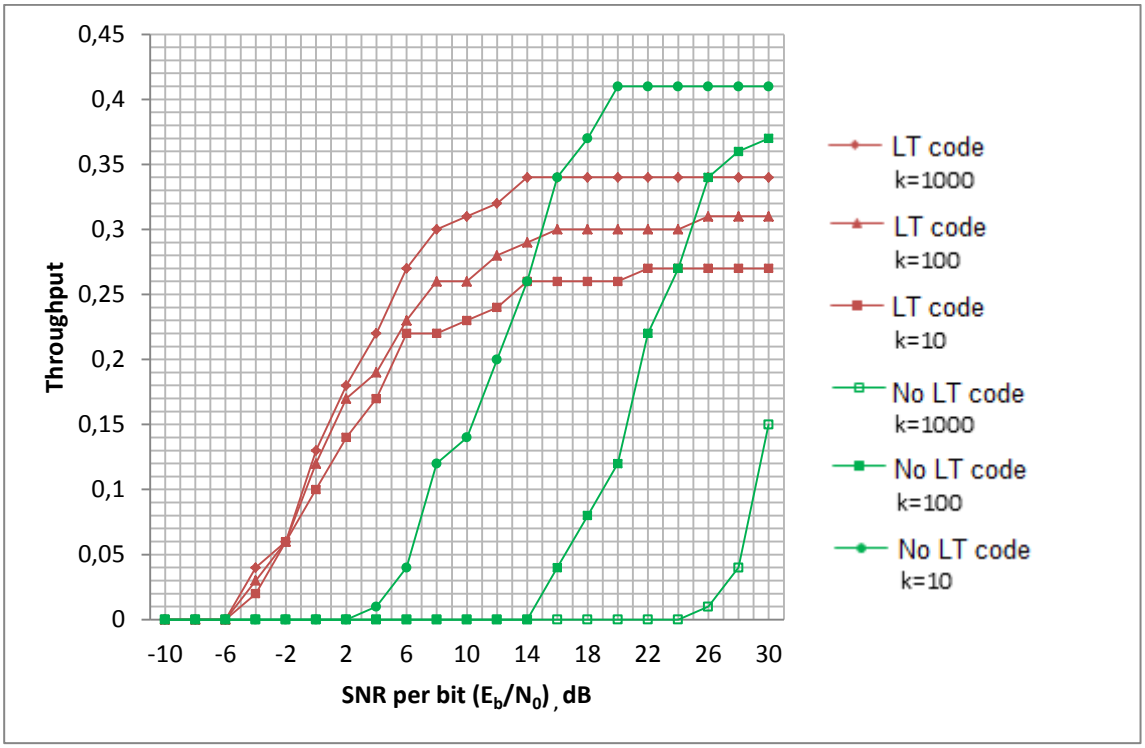


Figure 3.13: Throughput values vs SNR for point-to-point scenario with 'no LT', and 'LT with Robust Soliton Distribution' ($c = 0.1$ and $\delta = 0.5$) for 100-bit packets and different numbers of input packets: *k* = 10, 100 and 1000.

If throughput graphs of Fig. 3.13 are examined it can be seen that the throughput increases with increasing number of input packets. LT codes give better throughputs at low received SNR's but they are observed to become useless above a certain SNR value for each *k*.

So far the change of throughput with changing number of input packets and packet length has been studied. Although this study gives information about changing number of packets and packet lengths, it is not possible to predict the exact

behaviour of LT codes when a fixed length data is segmented in different ways. In the next section, this issue will be discussed.

3.7. SIMULATION RESULTS FOR THE SAME TOTAL DATA LENGTH AND DIFFERENT PACKET LENGTHS

As indicated in **[Wang, Chen, 2010]**, there is neither a closed form expression nor a general rule to segment data into different number of packets with different lengths efficiently in terms of throughput values. However, it can be roughly said that higher number of input packets (k) with shorter lengths (l) are more efficient for low SNR values, while it is more efficient to decrease k for high SNR's, by examining our results presented in Figure 3.14, obtained by averaging over 10 trials.

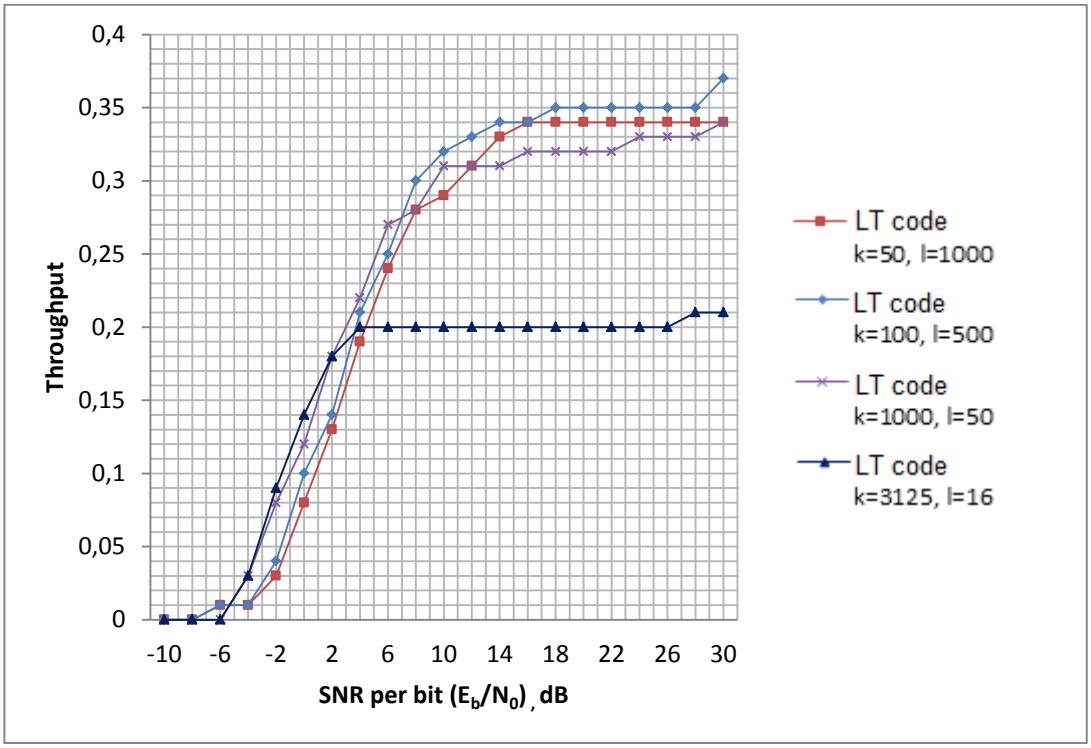


Figure 3.14: Throughput values vs SNR for point-to-point scenario with LT codes and Robust Soliton Distribution ($c = 0.1$ and $\delta = 0.5$) for different segmentations of 50000-bit data.

3.8. CHAPTER SUMMARY

In this chapter, we have studied the effects of degree distribution and data segmentation on throughput, when LT codes are used in point-to-point communication and compared them to ‘no LT codes’ case.

It is observed that to increase the throughput, δ must be increased and c must be decreased. These results agree with the results presented in **[Luby, 2002]** and **[MacKay, 2005]**.

Our observations related to data segmentation support those of **[Wang, Chen, 2010]**, and show that LT codes yield generally better throughputs than ‘no LT codes’ case for low SNR values. In addition, increasing the number of input packets without changing the symbol length increases the throughput regardless of the received SNR if it is higher than a certain value. On the other hand, increasing the symbol length without changing the number of input packets may affect the throughput in both directions depending on the received SNR; and the same conclusion is valid also for the segmentation of a fixed-size data. It can be generally said that it is better to use smaller symbol lengths to increase the throughput for low received SNR values.

CHAPTER 4

SIMULATION RESULTS FOR PROPOSED METHODS

After examining the characteristics of LT codes and comparing these codes with the cases where there is no LT code, as described in **[Wang, Chen, 2010]** by preliminary simulations, we have tried two ideas to improve the performance, *i*) periodic transmission of degree-one encoding symbols and *ii*) limitation of high-degrees.

Since the decoding of LT codes requires degree-one encoding symbols to start and to properly continue, we have considered that sending degree-one encoding symbols periodically might increase the throughput. This case is analyzed in Section 4.1; but observed to yield no significant improvement. As indicated in Chapter 3, some encoding symbols with unnecessarily high degrees are generated in the encoder. So, we have conjectured that preventing these high-degree encoding symbols from being transmitted might make LT codes more efficient. Simulation results obtained by blocking high-degree encoding symbols are presented in Section 4.2, but no appreciable refinement is achieved. Section 4.3 focuses on throughput change for packet erasure probabilities. The effect of removing the error correction mechanism of convolutional codes from the previously used system model is discussed in Section 4.4. In Section 4.5, a method proposed in **[Lu, Foh, Cai, Chia, 2013]** to continue the decoding even if Ripple is empty is discussed.

4.1. REPEATEDLY SENT DEGREE-ONE ENCODING SYMBOLS

To commence the decoding process or to continue it when Ripple is empty, a degree-one encoding symbol is required. If somehow degree-one encoding symbols are sent less frequently or erased, more packets will be required to be transmitted, which decreases the throughput. By intuition, it can be said that to check last few packets and to send a degree-one packet if no degree-one packet has been sent recently will increase the throughput. There is no research found in the literature for repeatedly sending degree-one encoding symbols.

We have performed simulations in which the encoder checks the last T transmitted encoding symbols. If there is no degree-one encoding symbols among these last T symbols, the encoder makes the degree of the next encoding symbol one. Symbol length, l, is chosen as 100 and number of input symbols, k, is chosen as 100 for the simulations.

As it can be seen in Figure 4.1 and 4.2, since the degree distribution is interfered by transmitting degree-one encoding symbols more frequently, there are more degree-one encoding symbols than the expected values depending on the Robust Soliton Distribution.

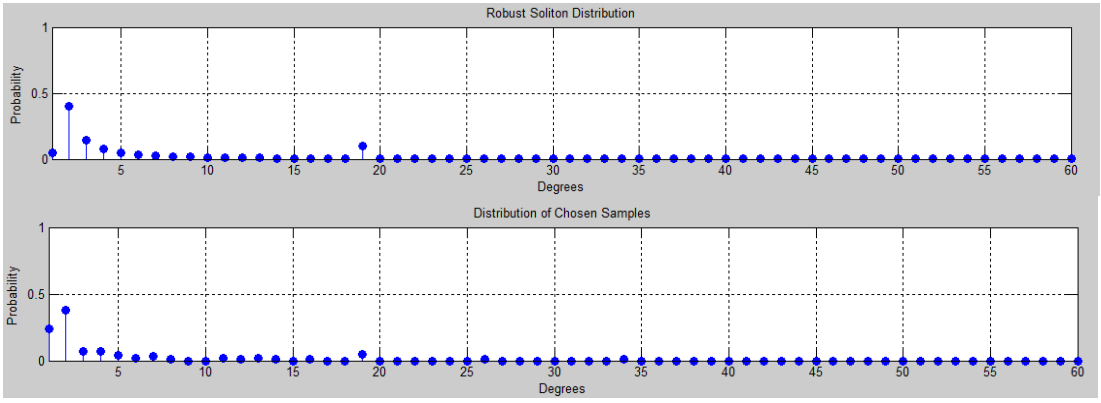


Figure 4.1: Comparison between sample degree distributions over 100 symbols (first 60 symbols are shown) according to repeatedly sending method and Robust Soliton Distribution ($c = 0.1$, $\delta = 0.5$ and T=5)

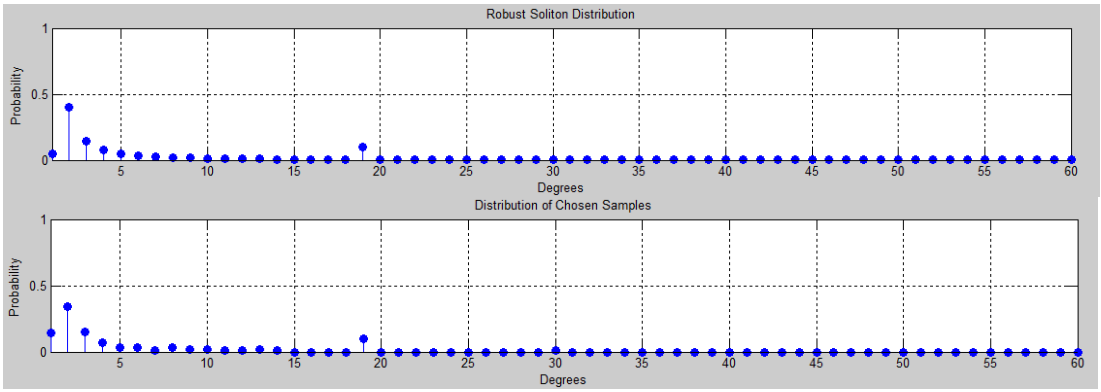


Figure 4.2: Comparison between sample degree distributions over 100 symbols (first 60 symbols are shown) according to repeatedly sending method and Robust Soliton Distribution ($c = 0.1$, $\delta = 0.5$ and T=10).

In Figure 4.3, throughput values for periodic repetition and no repetition of degree-one encoding symbols are given. Each point in this figure is obtained by averaging over 20 trials. Although periodic repetition of degree-one symbols gives better results for some SNR values, there is no significant improvement when compared to no repetition case. Simulation results do not seem to support our intuition about the advantage of sending degree-one encoding symbols repeatedly.

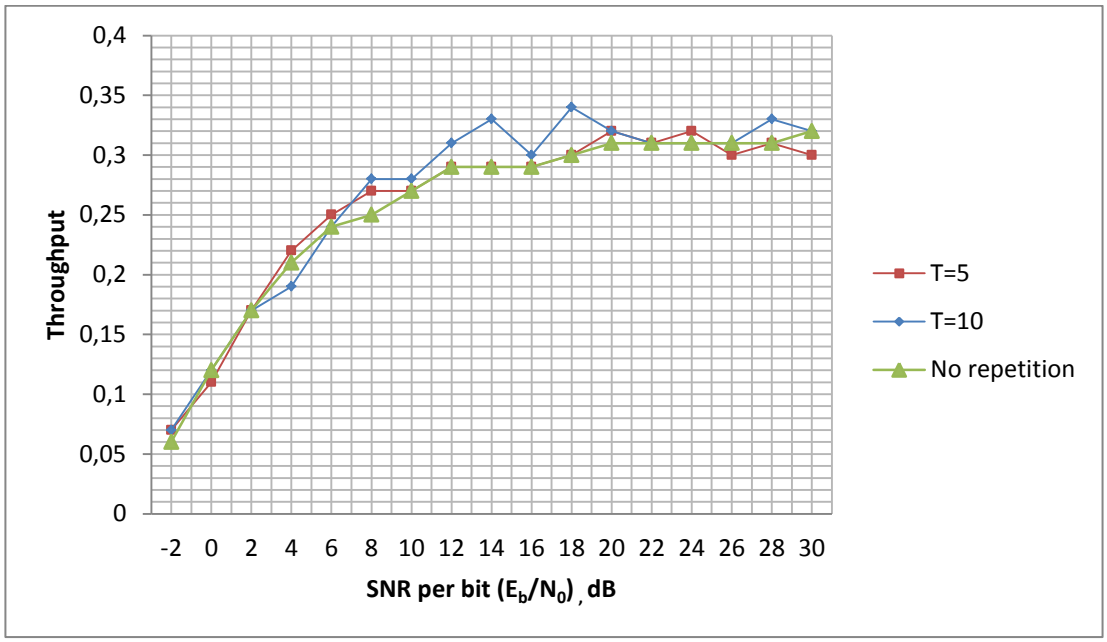


Figure 4.3: Throughput comparison between Robust Soliton Distribution (c=0.1, δ=0.5) and repeated sending of degree-one symbols with separation T=5 and 10, for point-to-point scenario.

4.2. ENCODING SYMBOLS WITH DEGREE LIMITATION

Although the Robust Soliton Distribution assigns high degrees very seldomly, except the degree corresponding to the second spike; still high degree encoding symbols can be observed among the transmitted packets. By intuition, it can be said that these high degree encoding symbols are useless and they decrease the throughput. Therefore, it is expected to obtain higher throughput values if the assigned degree is limited to a maximum value.

Simulations are performed as follows: A limit for maximum degree is defined and degrees of encoding symbols whose firstly assigned degrees are higher than this limit are reassigned according to a uniform distribution between 1 and this maximum value. Number of input symbols and the symbol length is chosen as 100, $c = 0.1$ and $\delta = 0.5$. Simulation results given in Figure 4.4 show that the limitation of the degrees does not yield higher throughput values.

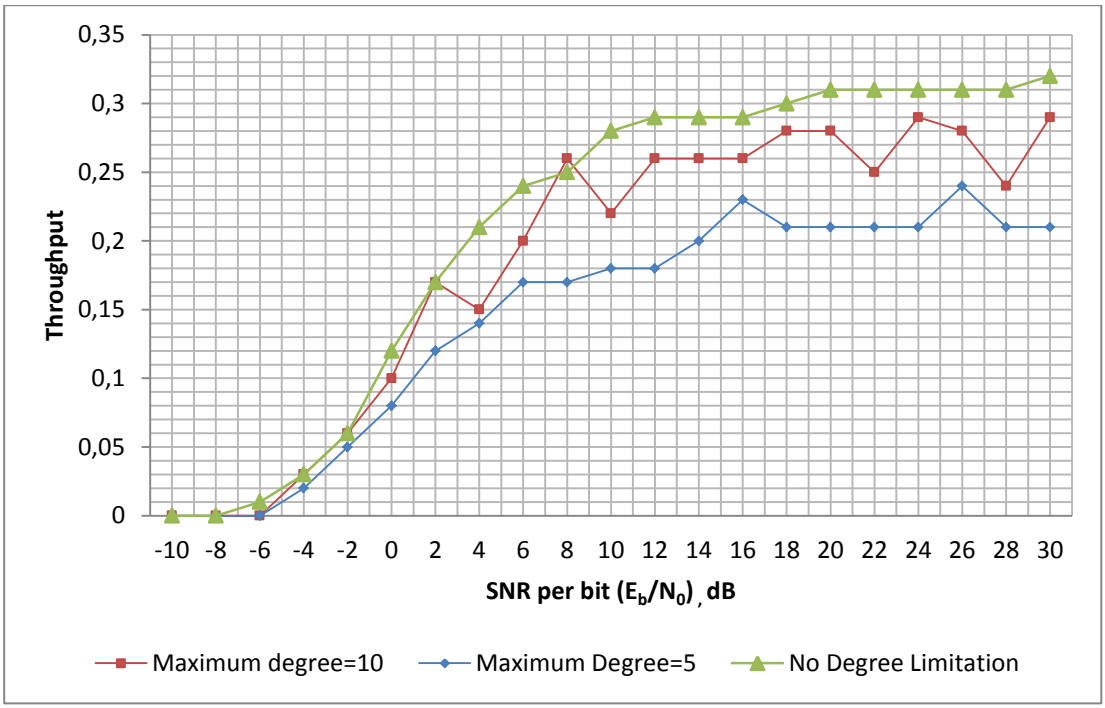


Figure 4.4: Throughput vs SNR values for Robust Soliton Distribution with maximum degrees limited to 5 and 10 (c=0.1, δ=0.5).

4.3. EFFECT OF PACKET ERASURES

Conditions such as noise, fading, interference, checksum errors, etc., might cause packets sent from the transmitter to the receiver to get lost or not to be decoded **[Yang, Shroff, 2012]**. Each packet might be erased with a probability of p_e, called the 'packet erasure probability'.

In this section, the effect of the packet erasure probability on the throughput is analyzed for the point-to-point scenario with and without the LT codes. 50000-bit data is sent by using 50 and 100 input symbols (packets). It is assumed that there are enough packets sent from the transmitter to the receiver to finalize the decoding successfully at the receiver in both cases, with and without LT codes. Decoder used for the 'no LT code' case is assumed to store and use the previously decoded packets in the succeeding transmissions.

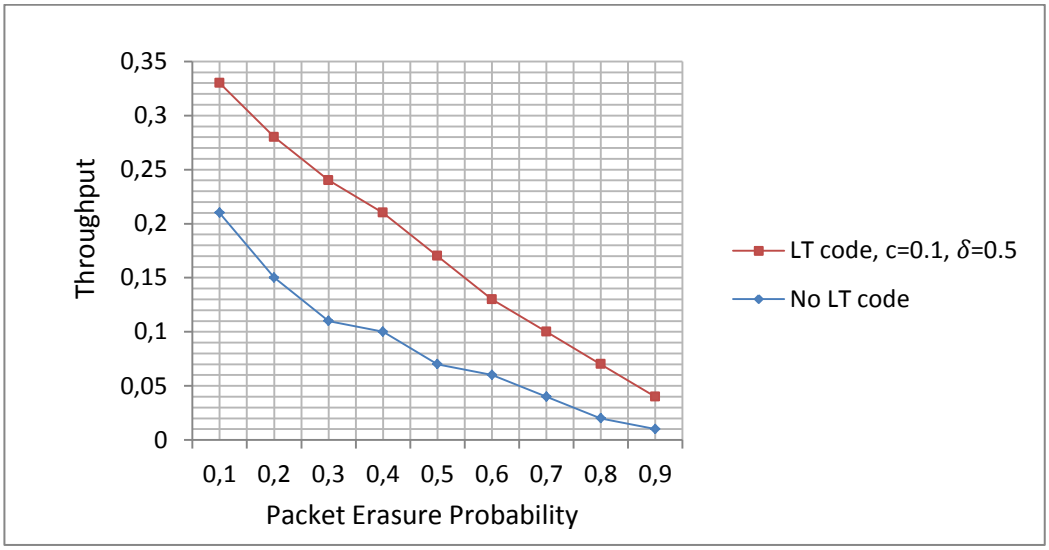


Figure 4.5: Throughput vs packet erasure probability for the point-to-point scenario with and without LT codes (number of packets=50, packet length=1000)

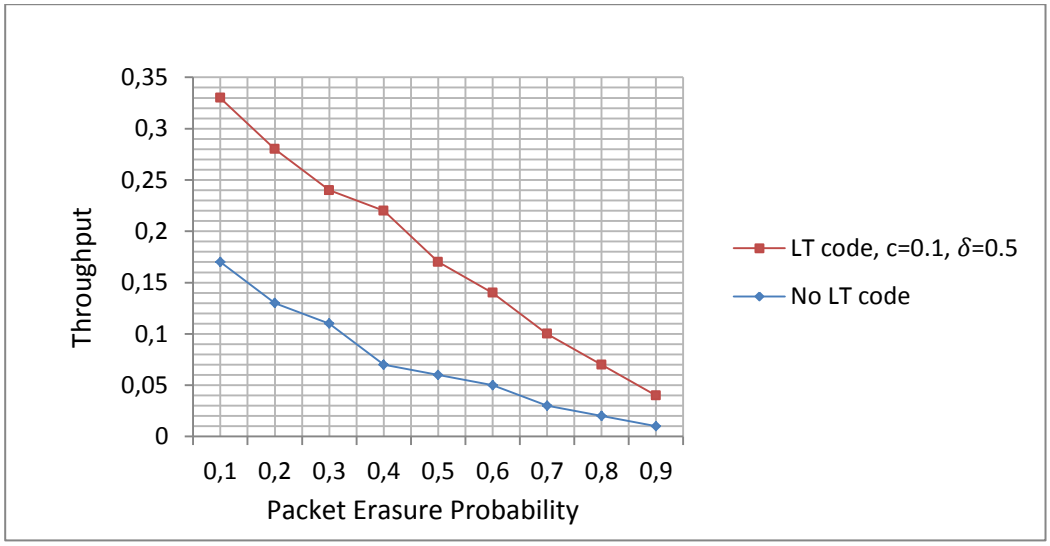


Figure 4.6: Throughput vs packet erasure probability for the point-to-point scenario with and without LT codes (number of packets=100, packet length=500)

As it can be seen from Figure 4.5 and Figure 4.6, throughput values of LT codes is higher than the throughput values of 'no LT codes' case for each packet erasure probability value. Performance of Luby Transform coding is not affected from the segmentation of the overall data into 50 or 100 packets, whereas the 'no LT codes' case is slightly better when the number of packets are smaller.

4.4. EFFECT OF USING CONVOLUTIONAL CODES WITH LT CODES

As previously indicated in the system model explanation, Luby Transform codes must be used with error detecting and/or correcting codes since they cannot detect or correct erroneously received encoding packets. In this case, the LT code can be considered as the ' outer code' while the CRC and the convolutional code can be thought as the 'inner codes'. In the decoding process of Luby Transform codes, it is assumed that all packets passed to the Luby Transform decoder are free of errors.

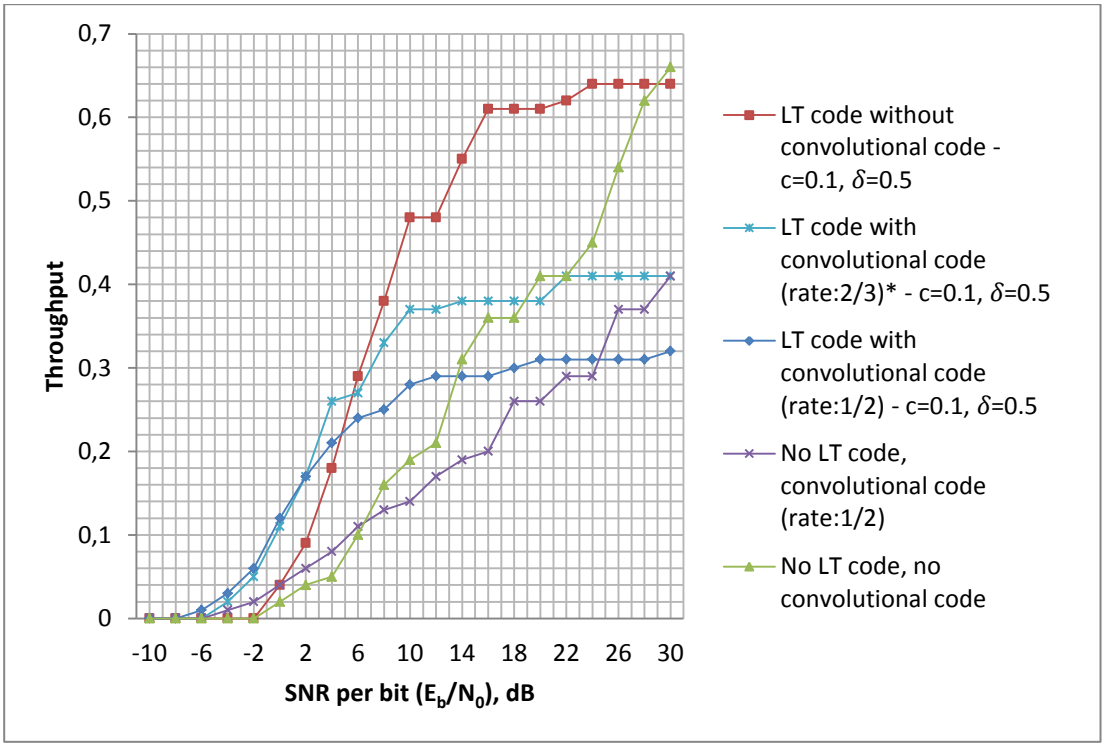
In the previous simulations, the CRC code is used to detect and the convolutional code is used to correct errors. To guarantee that all packets passed to the Luby Transform decoder are free of errors, erroneously received packets have to be detected before Luby Transform decoding. Therefore, using CRC or another error

detecting code is a must, unless it can be ensured that all packets received by the receivers are correct. This can be ensured only if the SNR value of the signal reaching the receiver is very high. Assuming that the SNR is always high is not a realistic approach.

On the other hand, using an "error correction" mechanism is not a must even if error correcting codes prevent packets from being erased. Without using the convolutional code, erroneously received packets can still be detected and erased. Therefore, by using only CRC and Luby Transform codes, it is possible to design an encoder and a decoder.

In this section, the effect of removing the convolutional code is analyzed and presented in Figure 4.7. For 'no LT code' simulations, it was assumed that all input packets are sent in an order in the first transmission and the following repetition of the whole data starts with the transmission of the first input packet again. In addition, it was assumed that the decoder can store the correctly received input packets obtained from the previous repetitions.

As it can be seen from Figure 4.7, throughputs for high SNR values are the best if neither LT nor convolutional codes are used. This is quite expected because if noise is not effective, there is no need for coding. However, for low SNR values, both codes are observed to increase the throughput; i.e., the use of both LT codes and the convolutional codes help. In order to choose the best curve among the five cases plotted in Figure 4.7, we compare the throughput performance of 'rate 2/3 convolutional code plus LT codes' to that of 'rate 1/2 convolutional code plus LT codes'; and observe that it is slightly worse at low SNR values, but much better at high SNR's. Hence, among the given curves, 'rate 2/3 convolutional code plus LT codes' seems to be the best choice for a broad range of SNR's.



*: *Rate* 2/3, *constraint length vector is* [5, 4] *and code generator matrix is* [23,35,0; 0,5,13]

Figure 4.7: Effects of using convolutional codes as an error correcting mechanism on throughput (number of packets=100, packet length=100) for the point-to-point scenario.

4.5. CASES WHEN RIPPLE IS EMPTY

In Luby's decoding algorithm, an encoding symbol whose degree is one must be received to start the decoding process. Since this symbol is a copy of one of the input symbols, after this specific input symbol is decoded, the decoding of its neighbors can proceed by XOR'ing this symbol with other encoding symbols. However, if Ripple gets empty before decoding all symbols, decoding cannot continue until a new degree-one encoding symbol is received. In **[Lu, Foh, Cai, Chia, 2013]**'s paper, this situation is assessed as the disadvantage of Luby's algorithm and it is analyzed to continue the decoding process when Ripple is empty.

In **[Lu, Foh, Cai, Chia, 2013]**, it is stated that an input symbol can be 'borrowed' when Ripple is empty. To decide which input symbol must be borrowed, neighbor list is analyzed and the input symbol which is carried by most packets is

chosen. Let b_j denote the index of this symbol. After borrowing this input symbol and decoding it, this symbol is passed to Ripple. After this step, decoding continues as Luby defines. Therefore, the algorithm is as follows:

if Ripple is empty
    compute the index of borrowed symbol b_j
    put index b_j into buffer
    recover the borrowed symbol
    release the borrowed symbol into Ripple
end

As indicated in **[Lu, Foh, Cai, Chia, 2013]**, decoding aims to solve

$$Mx = y$$

for x where x is the $k \times l$ matrix of the input symbols, y is the $n \times l$ matrix of received encoding symbols, l is the length of an input symbol, k is the number of input symbols, n is the number of received encoding symbols and M is the $n \times k$ coefficient matrix defined on $GF(2)$. Only one particular input symbol must be found. The first task is to identify a collection of row vectors from M such that the row vectors eliminate all other symbols except the borrowed symbol **[Lu, Foh, Cai, Chia, 2013]**. For this purpose, a vector, x′, can be defined to decide which rows of M must be used. If $x'_j = 1$, j^{th} row of M is used. Otherwise, $x'_j = 0$. By solving $M^T x' = u_i$, where u_i is a $k \times 1$ unit vector in which the unique 1 locates at the index i which is also the index of the borrowed symbol and assuming that $n = k$, x′ can be found. The inner product $< x', y >$ gives the borrowed symbol.

If M is not a full rank matrix, it cannot be guaranteed there is a solution and decoding fails. In case of $n > k$, an M_c matrix can be defined and x′ can be found by solving $M^T M_c x' = u_i$, where M_c is an $n \times k$ matrix and $M^T M_c$ is of full rank. The inner product $< M_c x', y >$ gives the borrowed symbol.

Depending on the obtained success ratio results in **[Lu, Foh, Cai, Chia, 2013]**'s paper, we expected to get higher throughput results from the simulations performed with full rank decoding algorithm. However, we could not obtain full rank M^T or $M^T M_c$ matrices frequently in our simulations. Therefore, our overall throughput results are almost the same as the cases, in which full rank decoding algorithm is not used.

4.6. CHAPTER SUMMARY

In this chapter, some proposals suggested by us or presented in literature were discussed. Firstly, we considered the case, where degree-one encoding symbols are sent repeatedly. This method yields an insignificant increase in throughput values. Secondly, we studied the case, where degrees of encoding symbols are limited; since our intuition says high degree encoding symbols are useless for decoding. We observed that degree limitation causes the characteristics of LT codes to become unstable and decrease the throughput. Then, we compared the throughputs of the cases with and without LT codes with respect to the the packet erasure probability. These results show that for the point-to-point scenario, LT codes give better throughputs than 'no LT code' case for the same values of the packet erasure probability. Segmentation of data does not make a difference in terms of the throughput for LT codes with respect to the packet erasure probability. We also studied the effects of rate 1/2 and rate 2/3 convolutional codes used with LT codes. Simulation results show that not only the LT codes but also the convolutional codes improve the throughput at low SNR's. If the received SNR is higher than a certain value, LT codes used with convolutional codes, give lower throughput results. Finally we discussed the proposal related to the case of empty Ripple suggested in the literature. In this proposal, initiating and continuing the decoding process of LT codes is evaluated as a drawback since the received encoding packets could have the sufficient information to decode another input symbol. However, we could not successfully implement this algorithm, since we could obtain full rank coefficient matrices (M^T or $M^T M_c$) matrices seldomly in our simulations.

CHAPTER 5

SIMULATION RESULTS FOR A PRACTICAL POINT-TO-MULTIPOINT TRANSMISSION SCENARIO

Although the simulation results show that Luby Transform coding technique is suitable for point-to-point transmission especially for low received SNR values, this technique can be evaluated as much more advantageous in case of point-to-multipoint transmission, where the same data is sent to several receivers from one transmitter.

There exist several cases where point-to-multipoint transmission is used in practical applications such as TV broadcast, satellite communication where one satellite sends the same data to several receivers, etc. One of the most important common criteria observed in these scenarios is that receivers do not send acknowledgments about whether they receive packets correctly or not since these scenarios are generally designed as one-way communication. Moreover, for some cases transmission power of receivers may not be sufficient to transmit a feedback signal, even if the power of signals sent by the transmitter is enough to reach the receivers.

To perform reasonable simulations, it is better to choose one practical application since parameters such as bandwidth, fading coefficients, transmission power etc. vary considerably depending on the scenario. In the simulations performed for point-to-multipoint transmission case, ‘firmware update of digital handheld frequency hopping radios used in public safety communication systems’ was chosen as the practical application.

In Section 5.1, our assumptions related to point-to-multipoint transmission system model are explained. In Section 5.2 and 5.3, simulation results for fixed and varying SNR values at each receiver in each trial are presented, respectively.

5.1. ASSUMPTIONS MADE FOR THE SYSTEM MODEL

The scenario for firmware update of digital handheld radios is as follows:

- Updating the firmware through the air is much more advantageous since radio must be connected to a PC having the necessary files otherwise. Therefore, this process is handled faster by using a practical method.
- Packet lengths used in the system of hand-held radios vary widely; however, average value can be accepted as 500-bit or 1000-bit.
- Total length for the firmware is about 300kB. However, this file consists of partial firmwares of different tasks. It can be accepted that there are approximately 50 different modules. Therefore, simulations can be performed for only one module, i.e. approximately 6kB data, for simplicity. (Simulator is capable of performing trials for larger data; however, results can be obtained faster when shorter data lengths are used. Therefore, 50000-bit data, which is approximately 6kB, was used for simulations.)
- Transmit power of the base station can increase up to 90W. However, transmit power of the handheld radios is only 5W. Therefore, if the radios are at the edge of coverage area of the base stations, the signals transmitted by these radios might not be able to reach the base station even if they receive the signals transmitted by base stations. This fact can be evaluated as ‘no acknowledgment’ case.
- 12.5kHz bandwidth is generally used for digital voice communication and 25kHz bandwidth is used for data communication. 25kHz is a narrow bandwidth when compared to the bandwidths used in 3G or LTE systems. Therefore, it is much more important to use the allocated frequency bands efficiently in public safety communication systems. Luby Transform Coding Technique is suitable for this purpose.

- Different modulation techniques such as C4FM, PSK, QAM, etc. can be used in public safety communication systems depending on the standards. However, BPSK was used in simulations for simplicity.
- Typical noise floor can be approximately accepted as −110dBm. Received signal power can be accepted as between −80dBm and −115dBm depending on the location where the radio is used. Therefore the maximum SNR value of received signals can be accepted as 30dB, and the minimum SNR can be considered as −5dB.
- SNR values vary widely for receivers depending on the distance to the base station. It is assumed that SNR values follow a uniform distribution with defined minimum and maximum values.
- Handheld radio users are generally stationary or they move slowly. Base stations are fixed. Therefore, slow fading can be assumed.
- There can be hundreds of users receiving signals from base station depending on the case. However, simulations are performed for 10 receivers for simplicity. An exemplary coverage area diagram for 10 receivers is given in Figure 5.1.

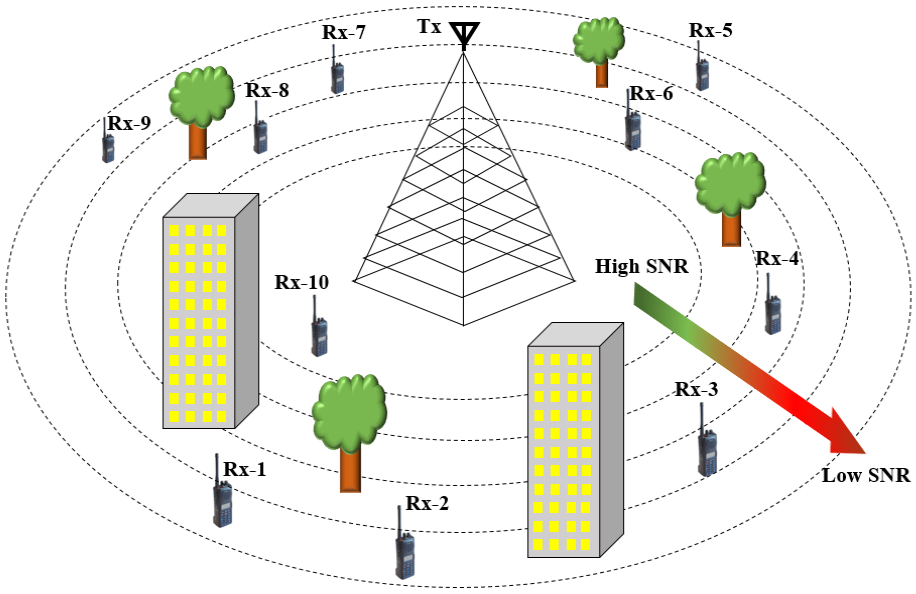


Figure 5.1: Exemplary coverage area diagram for 10 receivers

In the following sections, results of point-to-multipoint transmission simulations performed with the parameters decided according to the aforementioned assumptions are presented. As indicated in Section 3.1, the transmitter is assumed to transmit limited number of packets for point-to-multipoint transmission. In addition, it is assumed that after sending the certain amount of packets it stops transmitting. We defined this scenario as 'point-to-multipoint'. If LT codes are not utilized, all packets are transmitted in order and after being sent once, they are retransmitted beginning from the first packet. This case is called 'point-to-multipoint with no LT code'. Similarly, the number of packets is limited for the case 'point-to-multipoint with LT code'; however, packets are sent as random functions of input symbols instead of being sent in order.

It is aimed to obtain optimum parameters such as the ones used in Robust Soliton Distribution, packet length, number of packets, etc. to increase the number of receivers as much as possible that can decode sent packets by the transmitter. In each trial, an average success ratio is calculated by counting the number of all decoded packets for each receiver. Since SNR values of received signals for each receiver have a major effect on the success ratio of decoding, simulations are categorized into two classes in terms SNR distributions: Fixed or varying SNR values for each receiver in each trial.

5.2. SIMULATION RESULTS OF FIXED SNR VALUES FOR EACH RECEIVER IN EACH TRIAL

Since there are infinitely many combinations of SNR values of the signals received by receivers, some exemplary combinations must be chosen to make analyses. In the simulations performed, two SNR combinations each having 10 receivers are chosen as given in Table 5.1. The SNR values are obtained by using a uniform distribution (with minimum: −5dB and maximum: 30dB) and each receiver is assumed to have this fixed SNR value for all trials. In addition, it is assumed that the number of encoding packets sent by the transmitter can be at most five times larger than the number of input packets. In each case, Luby Transform coding is

compared with the cases without Luby Transform coding in terms of the decoding success ratio averaged over 10 receivers.

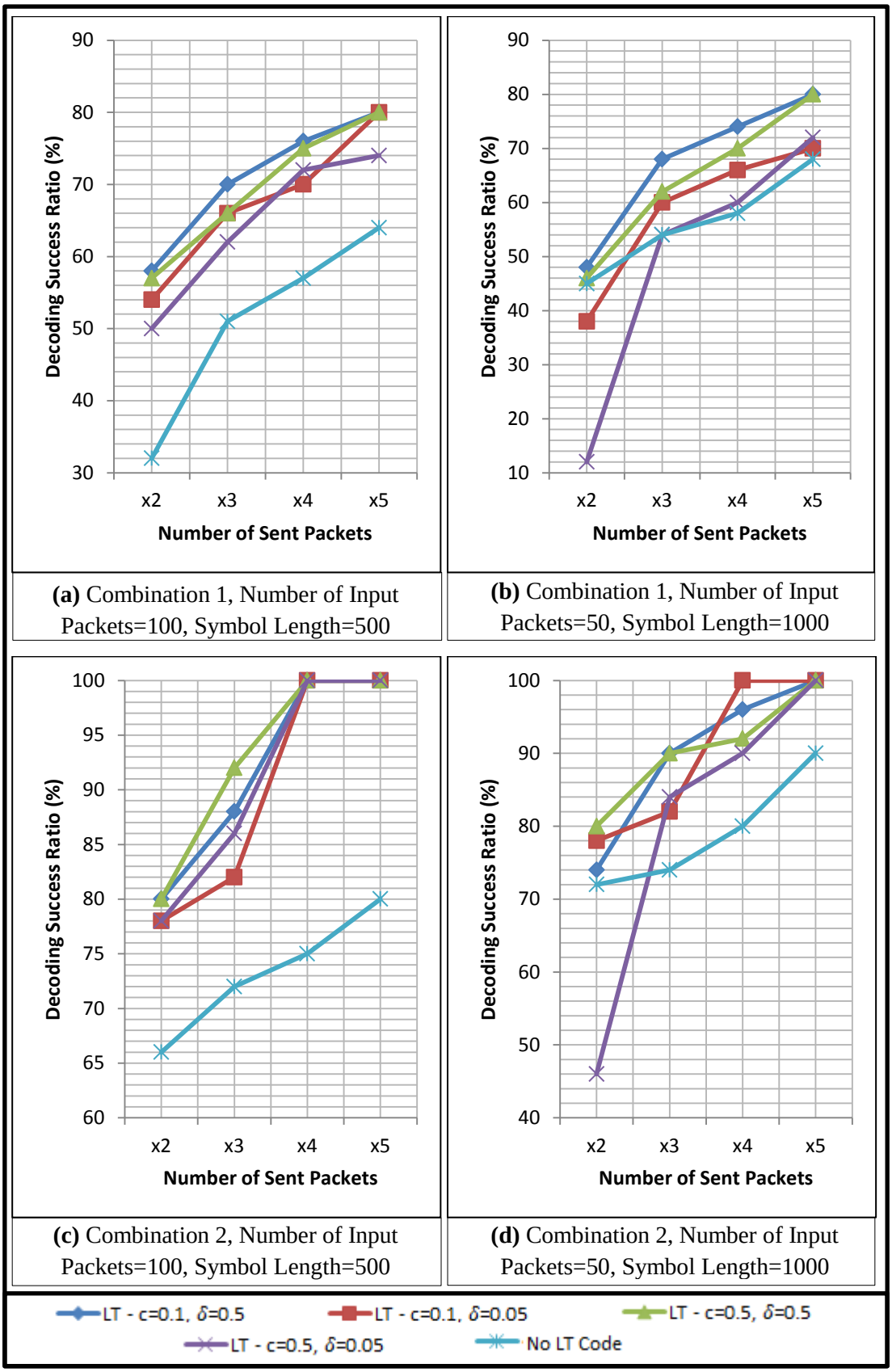
Table 5.1: Combinations related to received SNR values used in simulations

Receiver	Combination 1	Combination 2
1	17.6dB	6.96dB
2	−4.71dB	18.09dB
3	13.57dB	3.78dB
4	6.62dB	20.38dB
5	18.43dB	25.23dB
6	8.43dB	19.12dB
7	20.75dB	12.55dB
8	2.89dB	28.41dB
9	25.56dB	1.69dB
10	0.84dB	27.74dB

In terms of Robust Soliton Distribution parameters, four different combinations are analyzed:

- c=0.1 and δ=0.5
- c=0.1 and δ=0.05
- c=0.5 and δ=0.5
- c=0.5 and δ=0.05

The data length is chosen as 50000 bits and it is segmented into either 500-bit 100 packets or 1000-bit 50 packets. In Figure 5.2, we plot the decoding success ratio versus number of transmitted packets averaged over 10 successive trials. As can be seen from this figure, Luby Transform coding gives better results for almost all (c, δ). The maximum success ratios of LT codes reach to 76% and 100% for combinations 1 & 2 respectively, when the number of sent packets is four times the number of the input packets. On the other hand, when LT codes are not used, the success ratio can increase only up to 68% and 90% for combinations 1 & 2 respectively, even if number of sent packets is five times the number of input packets. As expected, the distribution of received SNR values has a major effect on the success ratio. The factor preventing the decoding success ratio of combination 1 from being 100% is the low received SNR at receivers 2 and 10.



(a) Combination 1, Number of Input Packets=100, Symbol Length=500

(b) Combination 1, Number of Input Packets=50, Symbol Length=1000

(c) Combination 2, Number of Input Packets=100, Symbol Length=500

(d) Combination 2, Number of Input Packets=50, Symbol Length=1000

Figure 5.2: Comparison of decoding success ratios of 'point-to-multipoint with no LT code' and 'point-to-multipoint with LT code' (fixed SNR values)

The combinations giving the highest decoding success ratio for almost all cases are (c=0.1, δ=0.5) and (c=0.5, δ=0.5) as can be seen in Figure 5.2. It is a consistent result with the ones observed for point-to-point transmission cases.

For Luby Transform coding, segmenting 50000-bit data into 100 packets (instead of 50) gives higher success ratios if the number of sent packets is limited (see Figure 5.2 for twice the number of input packets). However, for the the case without LT code, segmentation into 50 packets rather than 100 packets seems to yield a better decoding success ratio. This behavior is consistent with the previous results presented for point-to-point transmission case.

As it can be seen from Figure 5.2(c), 100% decoding success ratio is obtained if the number of sent packets is four times the number of input packets. In this case, data rate can be calculated as follows:

$$Symbol\ period \cong \frac{1}{Bandwidth} = \frac{1}{25\ kHz} = 0.00004\ seconds$$

$$Total\ time = \ Number\ of\ sent\ bits\ \times symbol\ period$$

$$= Number\ of\ sent\ packets \times packet\ length\ \times symbol\ period,$$

where packet length is equal to 2 × (symbol length + number of added bits for CRC + number of padded bits for convolutional coding). Then,

$$Total\ time = 400\ \cdot 1044\ \cdot 0.00004 = 16.704\ seconds$$

Therefore,

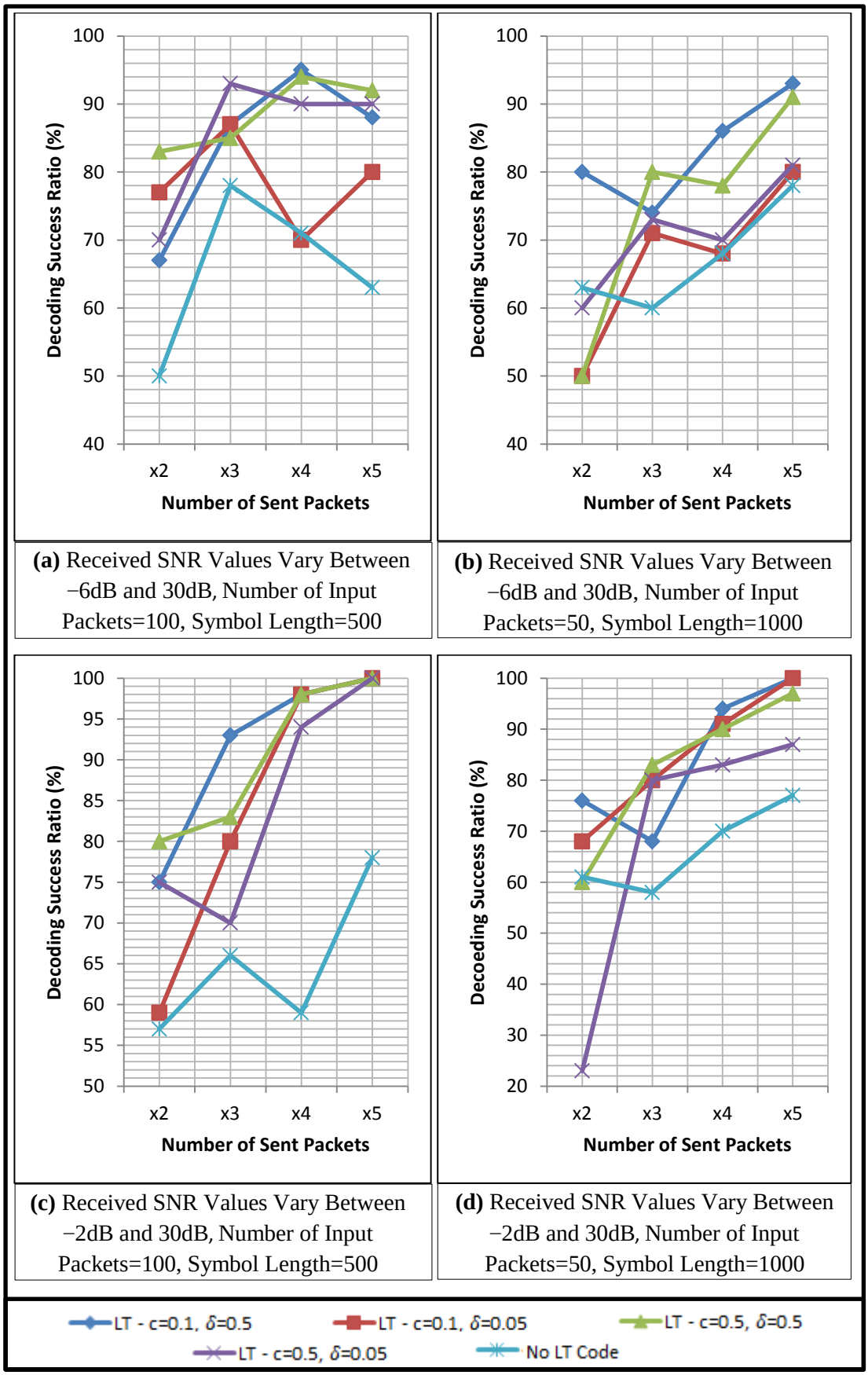
$$Data\ rate = \frac{Total\ data\ length}{Total\ time} = \frac{50000}{16.704} \cong 3kbps$$

Maximum decoding success ratio for ‘point-to-multipoint with no LT code’ presented in Figure 5.2(c) is 80%. This means that data rate is much lower than 2.39kbps, which would be the data rate of the no LT code case if decoding success ratio was 100%, when the number of sent packets is 500. These results show that the scenario using LT codes is at least 25% faster than ‘no LT code’ case.

5.3. SIMULATION RESULTS OF VARYING SNR VALUES FOR EACH RECEIVER IN EACH TRIAL

Instead of analyzing some specific combinations, making simulations by defining minimum SNR values only and allowing these values to vary in each trial is a useful method to comprehend how advantageous Luby Transform codes are when it is aimed to obtain a high decoding success ratio at all receivers. As indicated in the aforementioned explanations of the point-to-multipoint transmission scenario, transmitter does not have the information related to received SNR values at each receiver; however, coding parameters used in the transmitter can be arranged by aiming successful decoding only for receivers where SNR values are higher than a certain value. To simulate this fact, it is assumed that SNR value at each receiver is higher than the defined minimum value; however, SNR values at each receiver can change in each trial of the simulations. In addition, it is assumed that the number of encoding packets sent by the transmitter can be at most five times larger than the number of input packets. 10 successive trials are performed for each case.

Since SNR values at each receiver vary randomly in each trial, number of receivers at which SNR value is low can be high in some trials. Receivers with low SNR values decrease the overall decoding success ratio. For this reason, in some of the simulations, decoding success ratio might decrease even if the number of sent packets increases, as it can be seen from Figure 5.3(a), (b), (c), (d), Figure 5.4(e) and (f). Although the change of decoding success ratio seems irregular, these results still give some useful outcomes. First of all, it can be seen that decoding success ratio of point-to-multipoint scenario ‘with LT codes’ is higher than that of ‘without LT codes’ for most of the cases. Secondly, selecting Robust Soliton Distribution parameters as (c=0.1, δ=0.5) or (c=0.5, δ=0.5) is advantageous for the cases where the number of sent packets is limited and defined minimum received SNR value is low. However, the random distribution of received SNR values at each trial causes the other combinations to have higher decoding success ratios for some of the cases. Finally, it is observed that reaching to 100% decoding success ratio is possible when the SNR value at each receiver is higher than a certain value (2dB in our simulations) even if SNR changes randomly at each receiver in each trial.



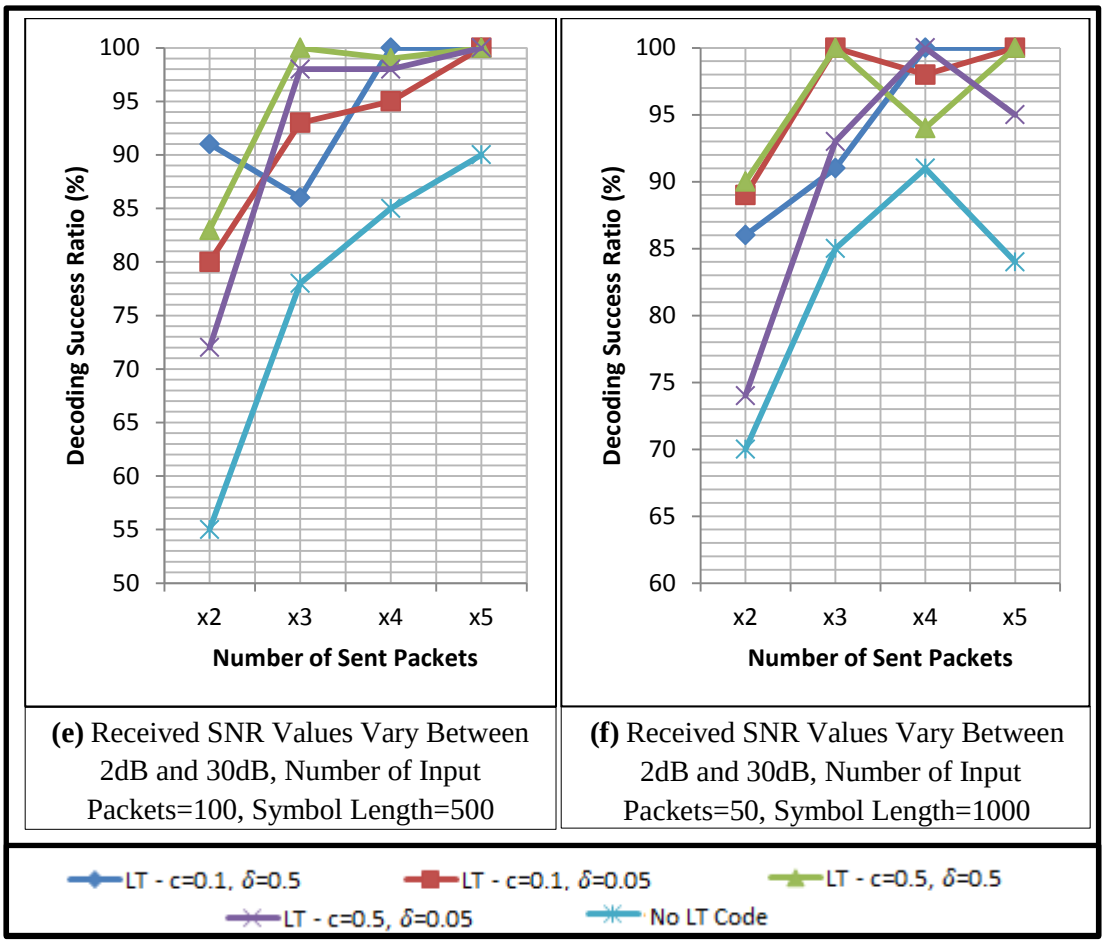
(a) Received SNR Values Vary Between −6dB and 30dB, Number of Input Packets=100, Symbol Length=500

(b) Received SNR Values Vary Between −6dB and 30dB, Number of Input Packets=50, Symbol Length=1000

(c) Received SNR Values Vary Between −2dB and 30dB, Number of Input Packets=100, Symbol Length=500

(d) Received SNR Values Vary Between −2dB and 30dB, Number of Input Packets=50, Symbol Length=1000

Figure 5.3: Comparison of decoding success ratios for the point-to-multipoint scenario with and without LT codes (varying SNR values)

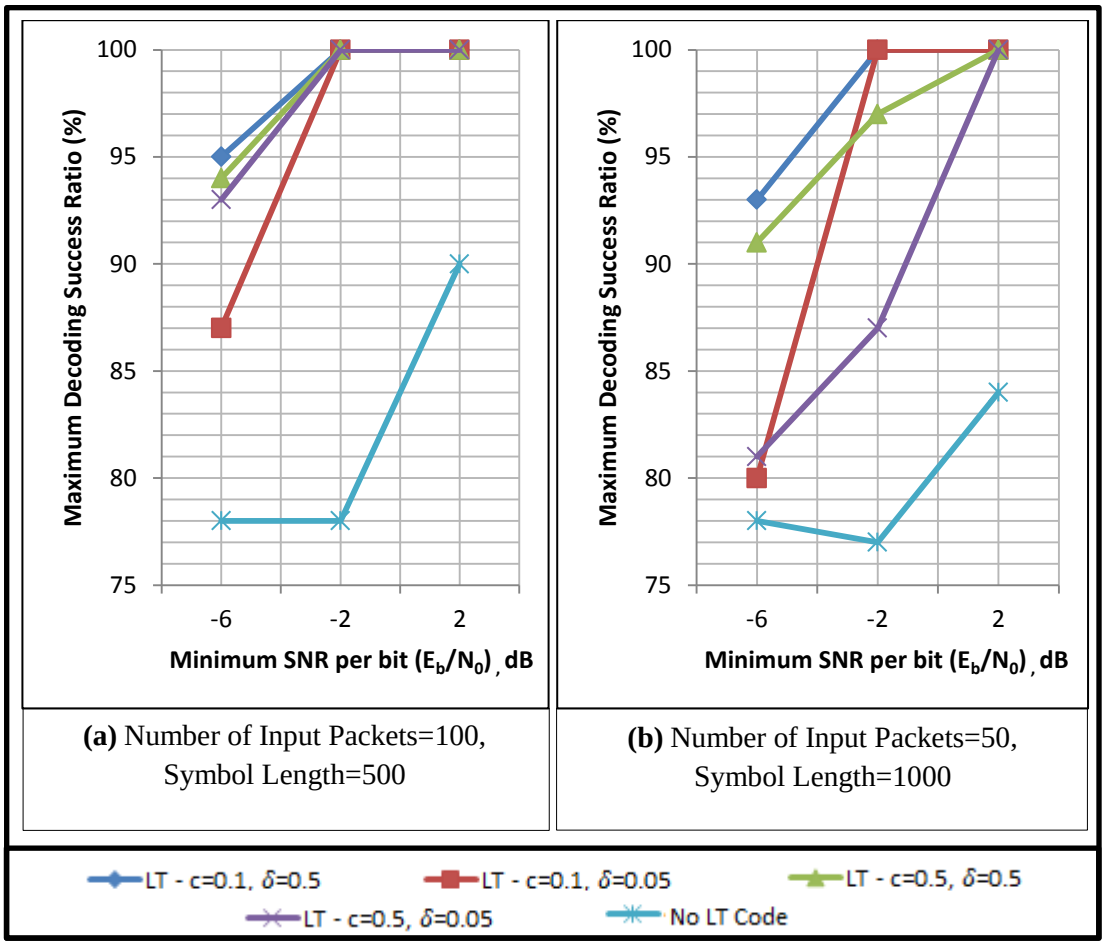


(e) Received SNR Values Vary Between 2dB and 30dB, Number of Input Packets=100, Symbol Length=500

(f) Received SNR Values Vary Between 2dB and 30dB, Number of Input Packets=50, Symbol Length=1000

Figure 5.4: Comparison of decoding success ratios for the point-to-multipoint scenario with and without LT codes (varying SNR Values, continued from Figure 5.3)

Figure 5.5(a) and (b) summarize the obtained overall maximum decoding success ratio results versus the defined minimum SNR values when the number of sent packets is at most five times the number of input packets. These results show that decoding success ratio of Luby Transform coding changes between 87% and 100% when defined minimum SNR values −6dB, −2dB and 2dB. However, decoding success ratio of the point-to-multipoint scenario without LT codes never reaches to 100% and can be around 90% at most.



(a) Number of Input Packets=100, Symbol Length=500

(b) Number of Input Packets=50, Symbol Length=1000

Figure 5.5: Comparison of the results obtained according to the defined minimum SNR values

5.4. CHAPTER SUMMARY

Although characteristics of LT codes are useful for point-to-multipoint transmission and this feature is emphasized in several papers, we could not encounter any research focusing on a point-to-multipoint transmission scenario. In this chapter, we presented a practical point-to-multipoint transmission scenario with our assumptions and simulation results comparing the cases with and without LT codes. Our results related to point-to-multipoint transmission case are consistent with the previously obtained findings related to the factors such as degree distribution parameters, number of input packets and symbol length. We performed our simulations for two main cases. In the first case, we assumed that all receivers have a constant received SNR for all trials. In the second case, which can be considered as an averaging over the first case, we assumed that the received SNR values of all receivers vary for all trials. Decoding success ratios for these cases showed no

appreciable difference. In addition, exemplary data rate calculations were performed for successfully decoded transmissions. Depending on the obtained results it can be said that using LT codes is quite advantageous to increase the possibility of decoding especially for the cases where there is no feedback mechanism, received SNR values are low and the number of encoding symbols to be sent is limited.

CHAPTER 6

CONCLUSIONS

Throughout this thesis, we have studied the use of Luby Transform codes in detail. LT codes might be much more preferable in the future for applications such as M2M communication, satellite communication, data storage, simultaneous software update, etc. In addition, Raptor codes, which benefit from Luby Transform codes, are planned to be used for Long Term Evolution. This also indicates the practicality of Luby Transform codes.

Research related to Luby Transform codes generally focus on Robust Soliton Distribution parameters, which can be assessed as one of the most important criteria affecting the efficiency of these codes. In this thesis, dependence of the system throughput on these parameters is studied.

Firstly, we have reviewed the theoretical approach to the distribution of encoding symbol degrees. All-At-Once and Ideal Soliton Distribution are studied. Then, encoding and decoding algorithms of Luby Transform codes are implemented in MATLAB.

Through simulations, which are performed for point-to-point communication in channels with noise and fading, Ideal Soliton Distribution is observed to be not a proper degree distribution, as Luby has also indicated **[Luby, 2002].**

Next, effects of Robust Soliton Distribution parameters are studied through simulations. Obtained results show that if δ, which is the bound on the probability of decoding failure, increases, throughput also increases. Besides, if c, which is a free parameter generally chosen as smaller than 1, decreases, throughput increases for the considered cases, where δ and c are chosen properly; i.e. the second spike can be observed and the ripple size is greater than 1.

Another factor having an effect on the throughput is the segmentation of the data. Simulation results show that increasing number of input packets generally increases the throughput. However, the packet length has different effects for low and high SNR cases. It is also observed that for low SNR values, Luby Transform codes have higher throughput values than the cases without LT codes as described in **[Wang, Chen, 2010]** and implemented in this work.

We have then tried two practical ideas which intiutively seem to work, although conflicting with the derivation of Robust Soliton distribution: degree limitation and repeated sending. However, simulation results did not yield any appreciable improvement, supporting the theoretical derivation of Luby that leads to the Robust Soliton Distribution.

In Chapter 5, a realistic scenario for point-to-multipoint transmission is presented and effects of some factors are analyzed. These simulations show that in most of the cases, especially when there are receivers with low received SNR and there is no acknowledgment mechanism between receivers and the transmitter, Luby Transform codes are much more advantageous than the cases not utilizing LT codes.

Although some specific cases were defined and simulations were performed according to these cases for both point-to-point and point-to-multipoint transmission through this thesis work, numerous practical scenarios can be analyzed by means of the simulator developed for our simulations. Developed simulator's structure is open to easy modifications in terms of channel conditions, data segmentation, degree distribution parameters and number of receivers.

As a future work, Luby Transform coding technique can be modelled as a Markov chain whose states are defined according to the combinations of XOR'ed input symbols for small message lengths as indicated in **[Hyytiä, Tirronen, Virtamo, 2007]**. Another open question is the case where there remains no degree-one symbol in the decoder. To continue Luby Transform decoding as defined by Luby, there always must be a degree-one encoding symbol to be able to send an input symbol to the ripple set. However, there is a method which is discussed in **[Lu, Foh, Cai, Chia, 2013]** and claimed to be more efficient since it is possible to continue decoding even if there is no degree-one encoding symbol at one of the steps

of decoding. Performing more detailed analyses on the method indicated in this paper could increase the throughput.

In literature, there are several researches focusing on degree distribution and decoding of Luby Transform coding technique. However, there are not so many analyses on the choice of neighbors. In **[Zhang, Hao, Xue, 2010]**, neighbor choice is analyzed in more detail; however, this analysis is made for the cases where a feedback message can be sent from the receiver to the transmitter, unlike the cases we studied where there is no feedback.

Energy efficiency is a significant reason for Luby Transform codes to be used in several communication areas, especially in wireless sensor networks where long-lasting batteries are required. There are several papers focusing on energy efficiency, such as **[Abouei, Brown, Plataniotis, Pasupathy, 2010]**. Since simulation results and articles in literature show that Luby Transform coding gives promising results for low SNR values, data can be transmitted with a lower transmission power by using Luby Transform codes. It seems like there will be a trade-off between energy efficiency and obtained throughput.

REFERENCES

[Abouei, Brown, Plataniotis, Pasupathy, 2010] Abouei, J., Brown, J., Plataniotis, K. and Pasupathy, S. (2010). On the energy efficiency of LT codes in proactive Wireless Sensor Networks. 1st ed. IEEE TRANSACTIONS ON SIGNAL PROCESSING. Available at: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=5472991

[Al-Ahmari, 2013] Al-Ahmari, A. (2013). Small Scale Fading and Multipath. In Wireless and Personal Communications (Lecture Notes). Available at:

http://ocw.kfupm.edu.sa/ocw_courses/user062/EE57701/Lecture%20Notes/EE%20577%20Lecture%207.pdf [Accessed 15 May. 2013]

[Belloni, 2004] Belloni, F. (2004). Fading Models. Available at: http://www.comlab.hut.fi/opetus/333/2004_2005_slides/Fading_models_text.pdf

[Botha, 2008] Botha, C. (2008). Designing Luby transform Codes As An Application Layer Reliability Mechanism for Media Streaming Over IP Network. Available at:

https://ujdigispace.uj.ac.za/bitstream/handle/10210/3702/Botha.pdf?sequence=1&isAllowed=y

[Fonseca, 2012] Fonseca, R. (2012). LT Erasure Codes (Programming Comps). Available at: http://cs.brown.edu/~rfonseca/comps/2012comps.pdf

[Goldsmith, 2005] Goldsmith, A. (2005). Statistical Multipath Channel Models. In Wireless Communications. Cambridge University Press.

[Hyytiä, Tirronen, Virtamo, 2007] Hyytiä, E., Tirronen, T. and Virtamo, J. (2007). Optimal Degree Distribution for LT Codes with Small Message Length. 1st ed. [ebook] Available at: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=4215906 [Accessed 3 Jul. 2013].

[Joshi, Rhim, Sun, Wang, 2010] Joshi, G., Rhim, J., Sun, J., & Wang, D. (2010). Fountain Codes (6.972 PRINCIPLES OF DIGITAL COMMUNICATION II). MIT.

[Lu, Foh, Cai, Chia, 2013] Lu, H., Lu, F., Cai, J. and Foh, C. (2013). LT-W: Improving LT Decoding With Wiedemann Solver. 1st ed. [ebook] IEEE TRANSACTIONS ON INFORMATION THEORY, VOL. 59, NO. 12. Available at: http://ieeexplore.ieee.org/xpl/articleDetails.jsp?tp=&arnumber=6609091&queryText%3Dlt+codes+decoding [Accessed 4 Jul. 2014].

[Luby, 2002] Luby, M. (2002). LT Codes. 1st ed. [ebook] Proceedings of the 43 rd Annual IEEE Symposium on Foundations of Computer Science (FOCS'02). Available at: http://ieeexplore.ieee.org/xpl/freeabs_all.jsp?arnumber=1181950 [Accessed 28 Mar. 2013].

[MacKay, 2005] MacKay, D. (2005). Fountain Codes. 1st ed. [ebook] Communications, IEE Proceedings- Volume:152, Issue: 6. Available at: http://ieeexplore.ieee.org/xpl/articleDetails.jsp?tp=&arnumber=1561992&queryText%3Dmackay+fountain [Accessed 12 May. 2013].

[MIT Ch-8, 2010] Chapter-8 Convolutional Coding. (2010). In MIT 6.02 Lecture Notes. MIT.

[MIT Ch-9, 2010] Chapter-9 Viterbi Decoding of Convolutional Codes. (2010). In MIT 6.02 Lecture Notes. MIT.

[Peterson, Brown, 1961] Peterson, P. and Brown, D. (1961). Cyclic Codes for Error Detection. 1st ed. [ebook] IEEE - Proceedings of the IRE (Volume:49 , Issue: 1). Available at: http://ieeexplore.ieee.org/xpl/articleDetails.jsp?tp=&arnumber=4066263&queryText%3DCyclic+Codes+for+Error+Detection [Accessed 19 Jul. 2014].

[Proakis, 2001] Proakis, J. (2001). Digital communications (4th ed.). Boston: McGraw-Hill.

[Sankar, 2014] Sankar, K. (2007). Bit Error Rate (BER) for BPSK modulation. Retrieved November 2, 2014. Available at:

http://www.dsplog.com/2007/08/05/bit-error-probability-for-bpsk-modulation/

[Tirronen, Virtamo, 2006] Tirronen, T., & Virtamo, J. (2006). Optimizing the Degree Distribution of LT Codes. HELSINKI UNIVERSITY OF TECHNOLOGY Department of Electrical and Communications Engineering Networking Laboratory.

[Wang, Chen, 2010] Wang, X. and Chen, W. (2010). Throughput-Efficient Rateless Coding with Packet Length Optimization for Practical Wireless Communication Systems. 1st ed. [ebook] Global Telecommunications Conference (GLOBECOM 2010), 2010 IEEE. Available at: http://ieeexplore.ieee.org/xpl/articleDetails.jsp?tp=&arnumber=5684359&queryText%3DXijun+Wang%2C+Wei+Chen [Accessed 13 Feb. 2014].

[Wesolowski, 2009] Wesolowski, K. (2009). Introduction to Digital Communication Systems. Chichester, West Sussex, U.K.: J. Wiley.

[Yang, Shroff, 2012] Yang, Y., & Shroff, N. (2012). Throughput of Rateless Codes over Broadcast Erasure Channels. Available at: http://maascom.ece.ohio-state.edu/publications/Shroff/ThroughputNetworkCoding.pdf

[Zhang, Hao, Xue, 2010] Zhang, W., Hao, C., & Xue, G. (2010). An Approach of Reducing Overhead in Luby Transform Codes. International Conference on Communications and Mobile Computing. Available at: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=5471346

APPENDIX A

CODING METHODS USED IN OUR SIMULATIOS

Cyclic Redundancy Check Generator

For Cyclic Redundancy Check coding technique, $n-k$ check bits are added to the k information bits. These binary digits can be considered as coefficients of a polynomial in the dummy variable X. For instance, a message 11011101 is represented as $1+X+X^3+X^4+X^5+X^7$.

Cyclic codes are defined according to a generator polynomial, $g(X)$. To encode a message polynomial $m(X)$, $X^{n-k}m(X)$ is divided by $g(X)$, remainder $r(X)$ is found and added to the $X^{n-k}m(X)$.

$n-k$ of low-order coefficients of resulting code polynomial are check bits. The rest of the coefficients of the resulting code polynomial is composed of information bits. **[Peterson, Brown, 1961]**

For example, $m(X)=11011101$ is encoded for $n=15$, $k=10$, $n-k=5$ and $g(X)=1+X^2+X^4+X^5$ as follows:

$$X^{n-k}m(X)=X^5(1+X+X^3+X^4+X^5+X^7)=q(X)g(X)+r(X)$$

where $q(X)$ is the quotient.

$$X^{n-k}m(X)=X^5+X^6+X^8+X^9+X^{10}+X^{12}$$

$$=(X+X^2+X^6+X^7)(1+X^2+X^4+X^5)+(X+X^2+X^3+X^4)$$

Since addition and subtraction are the same in modulo two arithmetic, code polynomial is

$$f(X)=X^{n-k}m(X)+r(X)=q(X)g(X)$$

$$f(X)=r(X)+X^{n-k}m(X)$$

$$f(X) = (X + X^2 + X^3 + X^4) + (X^5 + X^6 + X^8 + X^9 + X^{10} + X^{12})$$

0 1 1 1 1 1 1 0 1 1 1 0 1

Check bits Information bits

In our system model CRC-16-CCITT, which is one of the most commonly used polynomials in practical applications, was used. ($g(X) = 1 + X^5 + X^{12} + X^{16}$)

Convolutional Encoder

When convolutional codes are used, only parity bits computed from input message bits are sent. In convolutional encoder, *r* parity bits (*r*>1) are created with various subsets through a sliding window. Constraint length (in bits), *K*, is used as a measure of the size of the window. This window is slided forward by one bit at a time producing *r* parity bits and creating a coding rate of 1/*r*. In Figure A.1, an exemplary diagram of convolutional encoder is given **[MIT Ch-8, 2010]**.

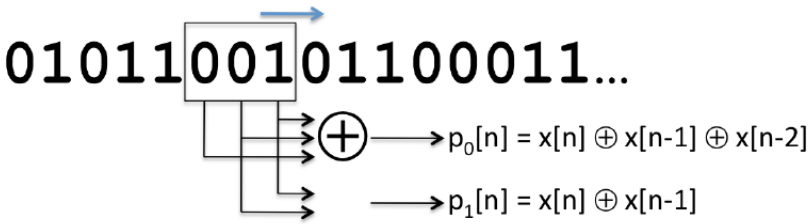


Figure A.1: Exemplary diagram of a convolutional encdoder

In the given example, constraint length is *K*=3 and *r*=2 parity bits are constructed for each input message bit according to the following equations:

$$p_0[n] = x[n] + x[n-1] + x[n-2]$$

$$p_1[n] = x[n] + x[n-1]$$

Moreover, it is assumed that *K*–1 '0' bits are padded in front to obtain proper initial conditions. **[MIT Ch-8, 2010]**

Parity bits are constructed according to a generator polynomial like in the previous equations. In this example, coefficients, g_i, of generator polynomial are (1,1,1) and (1,1,0). Generally, parity bit construction can be denoted as follows:

$$p_i = \left(\sum_{j=0}^{K-1} g_i[j]x[n-j]\right) mod 2, \text{ where } x[n] = 0 \; \forall n < 0.$$

For instance, let the input be 101100 ('00' sequence is padded in front in this case) where $g_0[0] = 1, g_0[1] = 1, g_0[2] = 1, g_1[0] = 1, g_1[1] = 1$ and $g_1[2] = 0$ as shown in block diagram given in Figure A.2. For these coefficients, parity bits are calculated as follows **[MIT Ch-8, 2010]**:

$p_0[0] = (1 + 0 + 0) = 1$

$p_1[0] = (1 + 0) = 1$

$p_0[1] = (0 + 1 + 0) = 1$

$p_1[1] = (0 + 1) = 1$

$p_0[2] = (1 + 0 + 1) = 0$

$p_1[2] = (1 + 0) = 1$

$p_0[3] = (1 + 1 + 0) = 0$

$p_1[3] = (1 + 1) = 0$

$p_0[4] = (0 + 1 + 1) = 0$

$p_1[4] = (0 + 1) = 1$

$p_0[5] = (0 + 0 + 1) = 1$

$p_1[5] = (0 + 0) = 0$

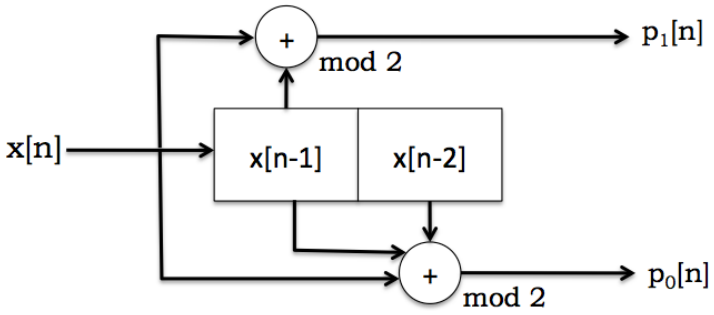


Figure A.2: Block diagram for $g_0[0] = 1, g_0[1] = 1, g_0[2] = 1, g_1[0] = 1, g_1[1] = 1$ and $g_1[2] = 0$

Another method used to view convolutional coding is state machine view. For the previous example, state machine view is given in Figure A.3. On the branches, produced parity bits are shown. For example, when the state is '00' and the next bit is '1', state changes to '10' and output becomes '11', which means both first and second parity bits are '1'. By using this state machine view, it can be seen that the output is 11 11 01 00 01 10 if the message is 101100 **[MIT Ch-8, 2010]**.

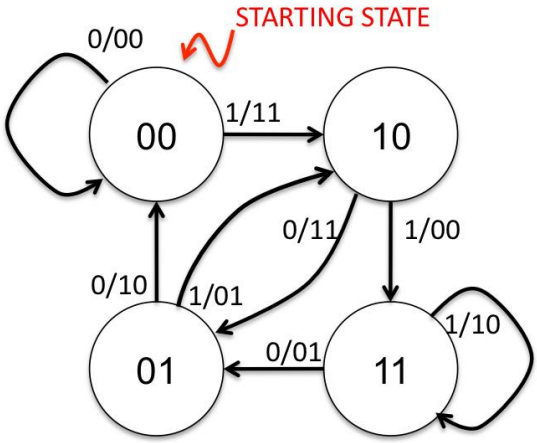


Figure A.3: State machine view for the previous example

$(r, k, K) = (2,1,7)$ convolutional code with generator polynomial $(171, 133)_8$ was used for the system model of our simulator (g_1=(1,1,1,1,0,0,1), g_2=(1,0,1,1,0,1,1)). This means that coding rate is $1/2$ and the number of padded '0' bits (K-1) is 6. These parameters for convolutional encoding is also commonly used for practical applications.

APPENDIX B

CHANNEL SIMULATION WITH NOISE AND FADING

Because of noise, fading, interference, etc., some of encoding packets cannot be decoded and cannot be used for Luby Transform decoding process even if error detection and correction techniques are used. These packets can be assessed as 'erased packets.' Therefore, the channel can be considered as a 'packet-erasure channel.'

To simulate the effects of a packet erasure channel, noise and fading effects were taken into consideration in our simulator.

Noise

Additive White Gaussian Noise (AWGN) is used for both in-phase and quadrature components in the system model. Therefore, the values of noise, *n*, follow the Gaussian distribution, whose probability density function is as follows:

$$p(n) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{\frac{-(n-\mu)^2}{2\sigma^2}} \quad \text{(B.1)}$$

where μ is mean and σ^2 is the variance. In simulations, μ=0 and σ^2=1. When BPSK is used, received signal becomes either $y = s_1 + n$ (if bit '1' is transmitted) or $y = s_0 + n$ (if bit '0' is transmitted), where $s_0 = +\sqrt{E_b}$ and $s_1 = -\sqrt{E_b}$. Then, conditional probabilities can be denoted as follows **[Proakis, 2001] [Sankar, 2014]**:

$$p(y|s_0) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{\frac{-(y-\sqrt{E_b})^2}{2\sigma^2}} \qquad p(y|s_1) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{\frac{-(y+\sqrt{E_b})^2}{2\sigma^2}} \quad \text{(B.2), (B.3)}$$

Conditional probability density funtion for BPSK is given in Figure B.1.

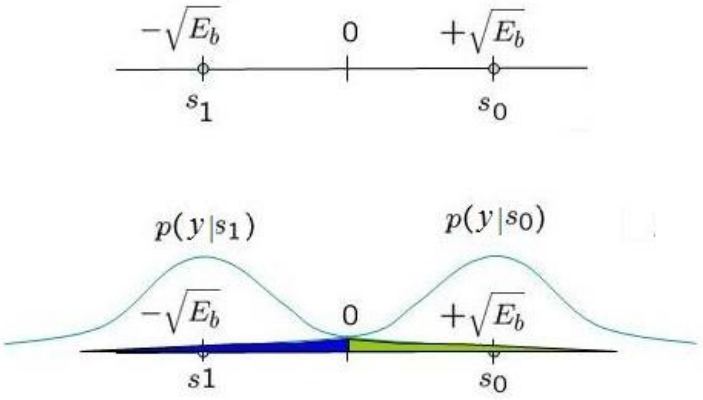


Figure B.1: Conditional probability density function for BPSK

Fading

Time and space are the factors affecting the channel between transmitters and receivers. Wireless communication channels can be evaluated as functions of time and space. Many replicas of the signal transmitted by the tranmitter reflect from different paths and reach to receiver; therefore, signal received by the receiver is combined of these replica. This is called the multipath effect. There are also cases that transmitter and receiver are not fixed. In these cases, movement causes propagation to vary along time. Therefore, fading occurs **[Belloni, 2004] [Goldsmith, 2005]**.

Movement of transmitter and receivers causes the carrier frequency to shift. Value of maximum Doppler frequency is a measure of this shift. The spectrum of the received signal broadens because of the Doppler effect. If this spread is narrow compared to the signal spectrum, the effect is not noticable. The channel where this effect is not noticable and whose channel characteristics are static is called a coherent channel. Coherence time is a measure of the time duration that channel characteristics do not change. The relation between maximum Doppler frequency and coherence time can be defined as follows

$$T_C \approx \frac{1}{f_D}$$

where T_C is the coherence time and f_D is the maximum Doppler frequency **[Goldsmith, 2005] [Al-Ahmari, 2013]**.

Another parameter used for characterization of wireless channels is the multipath spread of the channel, T_m, which can be considered as the measure of time between the earliest and latest multipath components **[Proakis, 2001]**.

$$B_C \approx \frac{1}{T_m}$$

where B_C is the coherence bandwidth. If a signal whose bandwidth, B_S, is much narrower than the coherence bandwidth, B_C, ($B_S << B_C$) is transmitted, fading can be assumed roughly the same across the entire signal bandwidth. This is called flat fading. Otherwise ($B_S \gg B_C$), the channel is frequency-selective. **[Goldsmith, 2005] [Proakis, 2001]**.

If the signaling interval, T_S, is much less than T_C, which means that channel impulse response changes more slowly than the transmitted baseband signal ($T_S << T_C$), slow fading occurs. It means that channel attenuation and phase shift are fixed for the duration of one signal interval. Otherwise ($T_S > T_C$), fast fading occurs **[Goldsmith, 2005] [Proakis, 2001]**.

In addition, for a flat and slowly fading, spread factor ($T_m f_d$) must be smaller than 1 **[Proakis, 2001]**.

$$T_m f_d < 1$$

If the impulse response of the channel is assumed to be zero-mean complex-valued Gaussian process, the envelope of the response at any instant is Rayleigh distributed and the channel is called Rayleigh fading channel **[Proakis, 2001]**.

Slow fading and Rayleigh Distribution were assumed to occur for our simulations. 'Rayleighchan' function of MATLAB was used. To obtain channel coefficients according to these assumptions, maximum Doppler frequency (f_D) was taken as 10 Hz and symbol period (T_S) was taken as 0.000001 for our preliminary simulations. It was also assumed that channel coefficients are known by the receiver.

APPENDIX C

SIMULATED RECEIVER (RX) COMPONENTS

BPSK Demodulator and Viterbi Decoder

Viterbi decoder is used to decode the parity bits constructed by convolutional encoding.

Suppose that the message is 101100 and parity bits are 11 11 01 00 01 10 as in the example given in the 'Convolutional Encoder' section. This can be shown via a Trellis diagram as in Figure C.1. **[MIT Ch-9, 2010]**

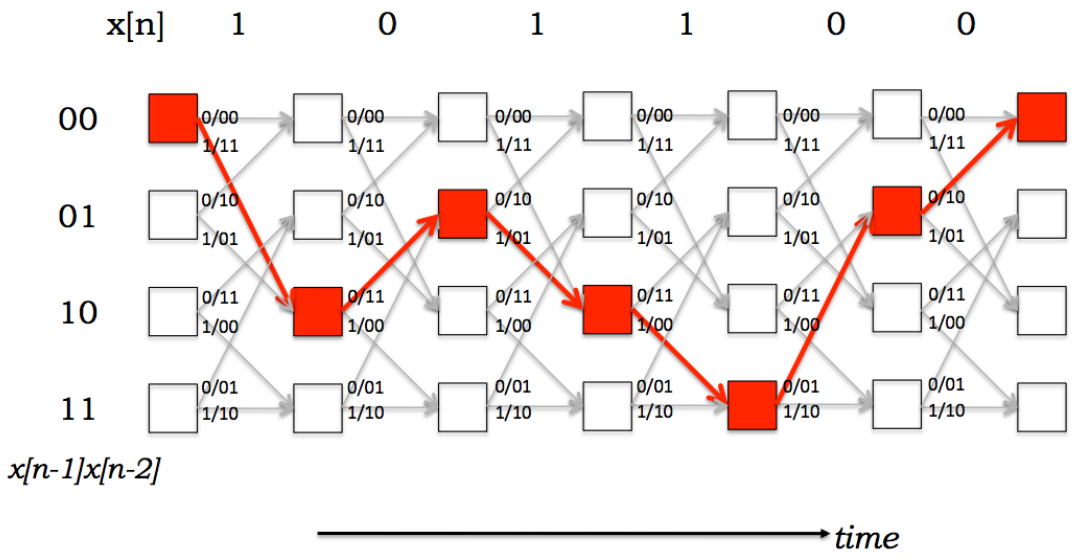


Figure C.1: Trellis diagram for construction of 11 11 01 00 01 10 sequence

Thanks to Viterbi decoding, parity bits created by convolutional encoders can be decoded without enumerating all possible combinations of message bits. This is performed via Trellis diagram. To decode an encoded sequence by using Trellis diagram, branch and path metrics are calculated. Hard decision decoding uses hamming distance to calculate these parameters while soft decision decoding uses euclidean distance by using the real voltage values without digitization.

For hard decision decoding, branch metric is calculated by comparing the received sequence and the possible sequences that can be constructed between the current state and next state. For instance, in Figure C.2, if the received sequence for the first time interval is '11', branch metric of the first branch transiting from 00 state to 00 state is 2 since the expected sequence is '00' while received sequence is '11'. However, branch metric of the second branch transiting from 00 state to 10 state is 0 since both the expected and received sequence are '11' in this case.

For hard decision decoding, path metric is calculated by adding the branch metric to the path metric of the previous state in the path. It is assumed that '00'state has the cost of 0 initially while other states have the cost of infinity. After calculating all the path metrics of surviving paths, path having the minimum path metric is chosen since it is the most likely path. By using this path, message sequence can be obtained and errors are corrected. In Figure C.2 and Figure C.3, received sequence is 11 10 11 00 01 10 while the output of the convolutional encoder is 11 11 01 00 01 10 (since the message is 101100). In these figures, branch metrics calculated according to hard decision decoding are shown on the branches and path metrics are shown in the boxes. After some steps, only surviving paths are drawn. At the bottom of Figure C.3, only the most likely path is drawn and message is decoded correctly **[MIT Ch-9, 2010]**.

For soft decision decoding, brench metrics and path metrics are calculated according to the voltage values of the signals to be sent as the input to the decoder. Generally, square of the difference between the expected voltage and recieved voltage values is used in practical applications. In this case, branch metric is calculated as follows **[MIT Ch-9, 2010]**:

$$Branch\ metric_{soft}[u, v] = \sum_{i=1}^{p} (u_i - v_i)^2$$

where u_i values are expected voltage values for p parity bits, each of which is 0 or 1, and v_i values are the corresponding received voltage values.

Viterbi decoding was implemented in MATLAB by using ‘vitdec’ function. According to decision type parameter, this function can be classified into three categories: unquantized, hard decision and soft decision. From these categories, ‘unquantized’ was used in this system model. It accepts real numbers as input and interprets positive real numbers as logical zero and negative real numbers as logical one, which also can be assessed as BPSK demodulation and soft decision is used together. It uses euclidean distance unlike hard decision decoding using hamming distance.

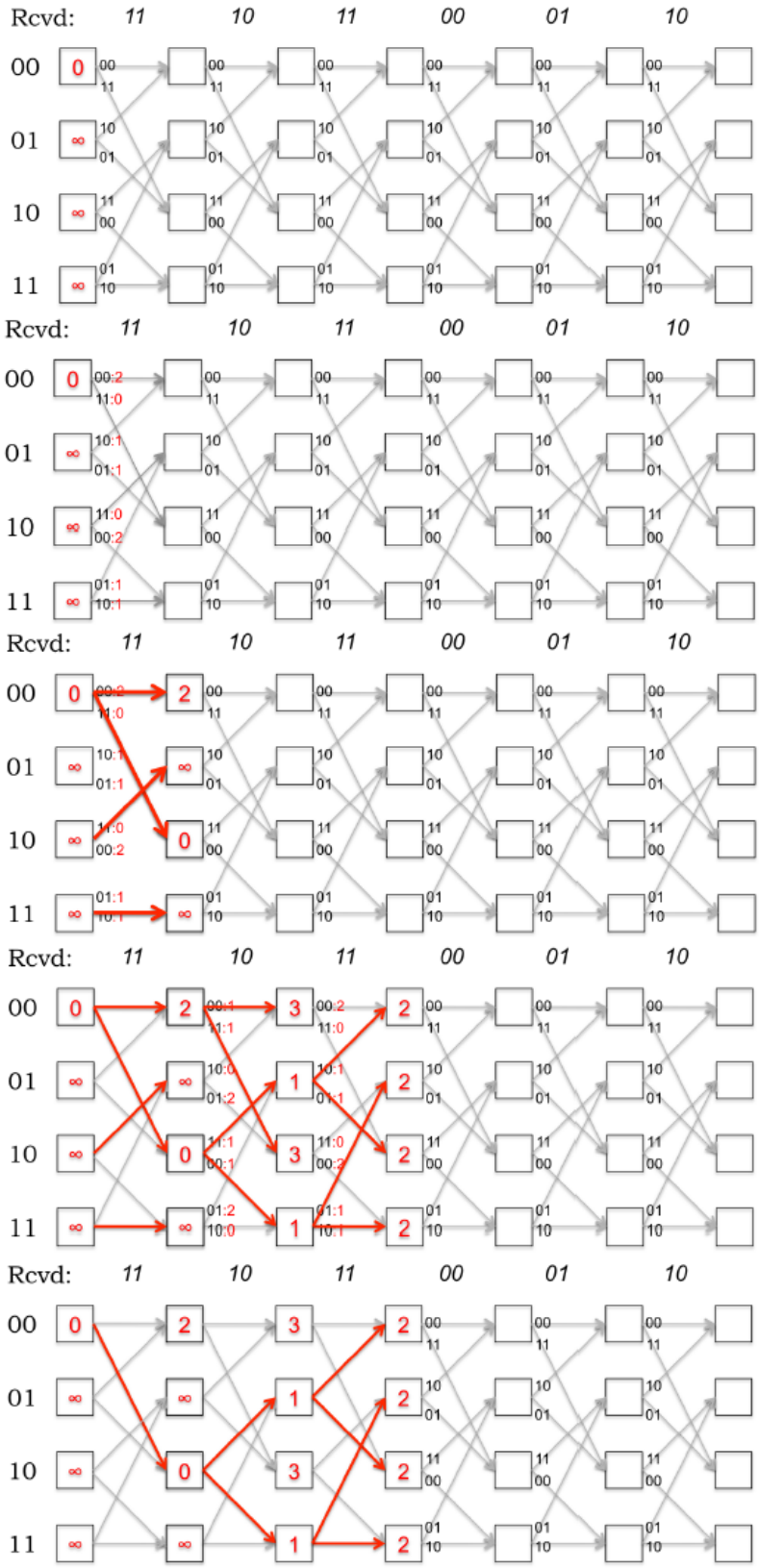


Figure C.2: Viterbi decoding for received sequence 11 10 11 00 01 10 (Reproduced from **[MIT Ch-9, 2010]**)

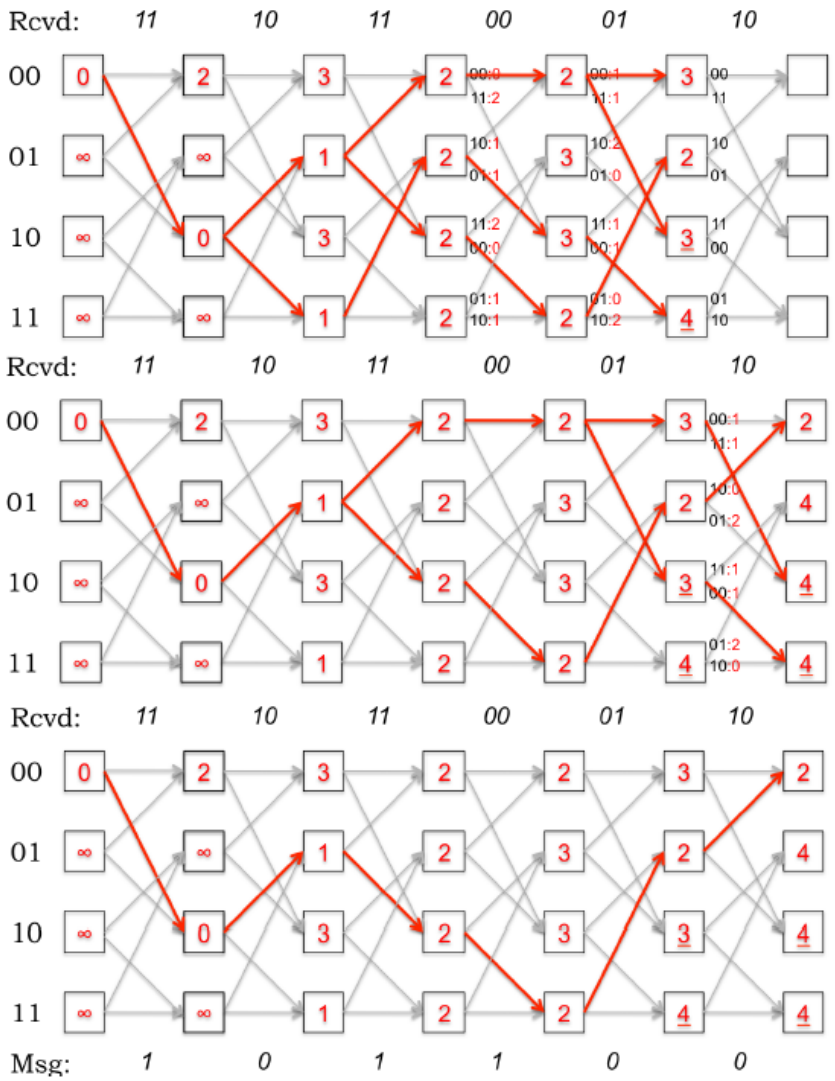


Figure C.3: Viterbi decoding for received sequence 11 10 11 00 01 10 (Continued from Figure C.2) (Reproduced from **[MIT Ch-9, 2010]**)

Cyclic Redundancy Check Detector

To detect whether there are any errors of packets coming from Viterbi Decoder, Cyclic Redundancy Check Detector is used in this system model. Obviously, it is designed according to CRC-16-CCITT, which is used in the encoder as CRC generator.

LT Decoder

LT Decoding algorithm is explained in the previous chapters. If an encoding packet is sent to LT Decoder from cyclic redundancy check detector, i.e. it is received or corrected properly, this encoding packet is decoded according to the Luby Transform decoding algorithm. A degree-one encoding symbol must be received by the decoder to commence decoding process. In the system model, it assumed that neighbors and degrees of encoding symbols are known by the decoder.

Reassembly

After all LT encoding packets are decoded (i.e. all input packets are found), these packets are reassambled and whole data is obtained.

APPENDIX D

ROBUST SOLITON DISTRIBUTION PARAMETERS AND DEGREE ASSIGNMENTS

As mentioned before, Robust Soliton Distribution, $\mu(i)$, is obtained from the Ideal Soliton Distribution $\rho(i)$, by adding $\tau(i)$,

$$\tau(i) = \begin{cases} \frac{R}{ik} \; for \; i = 1,2, \dots, \frac{k}{R} - 1 \\ \frac{R}{k} ln\left(\frac{R}{\delta}\right) \qquad for \; i = \frac{k}{R} \\ 0 \; for \quad i = \frac{k}{R} + 1, \dots, k \end{cases}$$

with a suitable normalization factor $\beta = \sum_{i=1}^{k} [\rho(i) + \tau(i)]$, as $\mu(i) = \frac{\rho(i)+\tau(i)}{\beta}$.

These definitions are also given in (2.32)-(2.34), where the expected Ripple size is $R = c \cdot \ln\left(\frac{k}{\delta}\right)\sqrt{k}$ for some constant $c > 0$, k is the number of input symbols, and δ is the allowable failure probability of the decoder. Definition of $\tau(i)$ at point $i = \frac{k}{R}$ causes the second spike in probability mass function plot. Therefore, Ripple neither vanishes nor grows too large before decoding ends. Because of this second spike, Robust Soliton Distribution gives more stable and efficient results for practical applications. In this appendix, our aim is to compare Ideal and Robust Soliton distributions numerically for different values of c and δ, keeping the number of input symbols fixed at $k = 100$.

In all the 3-part figures given below, the first part is the Ideal Soliton distribution $\rho(i)$, the second part is the additional term $\tau(i)$ and the third part corresponds to the Robust Soliton distribution $\mu(i)$. The spike of the Ideal Soliton distribution is at $i = 2$, and the Robust Soliton distribution has an additional second spike at $i = \frac{k}{R}$, that changes with parameters k, c and δ; since $R = c \cdot \ln\left(\frac{k}{\delta}\right)\sqrt{k}$.

For Robust Soliton Distribution, it is important to choose the values of c and δ properly. In **[MacKay, 2005]**, there is an analysis on number of degree-one encoding packets and total number of required encoding packets for different c and δ values regardless of received SNR; i.e., it is assumed that all packets are received correctly. Depending on the results presented in Chapter 3, provided that the expected Ripple size is chosen reasonably (in contrast to our last choice in this appendix), it can be generally said that throughput increases when δ (delta) increases and c decreases.

So, we order the below cases in the increasing order of expected throughputs. Cases 1 & 2 are for $\delta = 0.05$ and the remaining cases are for $\delta = 0.5$. For constant δ, we decrease c in successive cases since it is expected to improve the throughput; however, the fifth case shows that there is a lower limit on c so that the expected ripple size R remains greater than 1. In other words, in order to have a performance improvement with Robust Soliton Distribution over the Ideal Soliton, there should be a second spike, at a degree less than the number of input symbols; which also implies an expected Ripple size greater than 1; i.e.,

$$\frac{k}{R} < k \;\Rightarrow\; R>1 \;\Rightarrow\; c \cdot \ln\left(\frac{k}{\delta}\right)\sqrt{k} > 1 \;\Rightarrow\; c > 1/\ln\left(\frac{k}{\delta}\right)\sqrt{k}\,.$$

So, for k =100 input symbols used in this Appendix,

$$\delta = 0.05 \;\Rightarrow\; c > 1/\ln\left(\frac{100}{0.05}\right)\sqrt{100} = 0.013,$$

$$\delta = 0.5 \;\Rightarrow\; c > 1/\ln\left(\frac{100}{0.5}\right)\sqrt{100} = 0.019 \text{ are the lower limits on } c.$$

If k =1000, these lower limits change as

$$\delta = 0.05 \Rightarrow c > 0.003, \;\; \delta = 0.5 \Rightarrow c > 0.004.$$

For k =10000, they become

$$\delta = 0.05 \;\Rightarrow\; c > 0.0008, \text{ and } \delta = 0.5 \;\Rightarrow\; c > 0.001.$$

In the first four cases given below, these lower limits on c are taken into account; but the last case is an example of how Robust Soliton distribution becomes similar to the ideal one when the lower limit on c is disregarded.

1) For $k = 100,\ c = 0.5$ and $\delta = 0.05$, $\mu(i)$

has two spikes at $i = 2$ (because of $\rho(i)$) and $i = \frac{100}{0.5 \cdot \ln\left(\frac{100}{0.05}\right)\sqrt{100}} \cong 3$ (because of $\tau(i)$).

In this case the expected Ripple size is R=$0.5 \cdot \ln\left(\frac{100}{0.05}\right)\sqrt{100} = 38$. In Figure D.1, plots of $\rho(i)$, $\tau(i)$ and $\mu(i)$ are given, respectively.

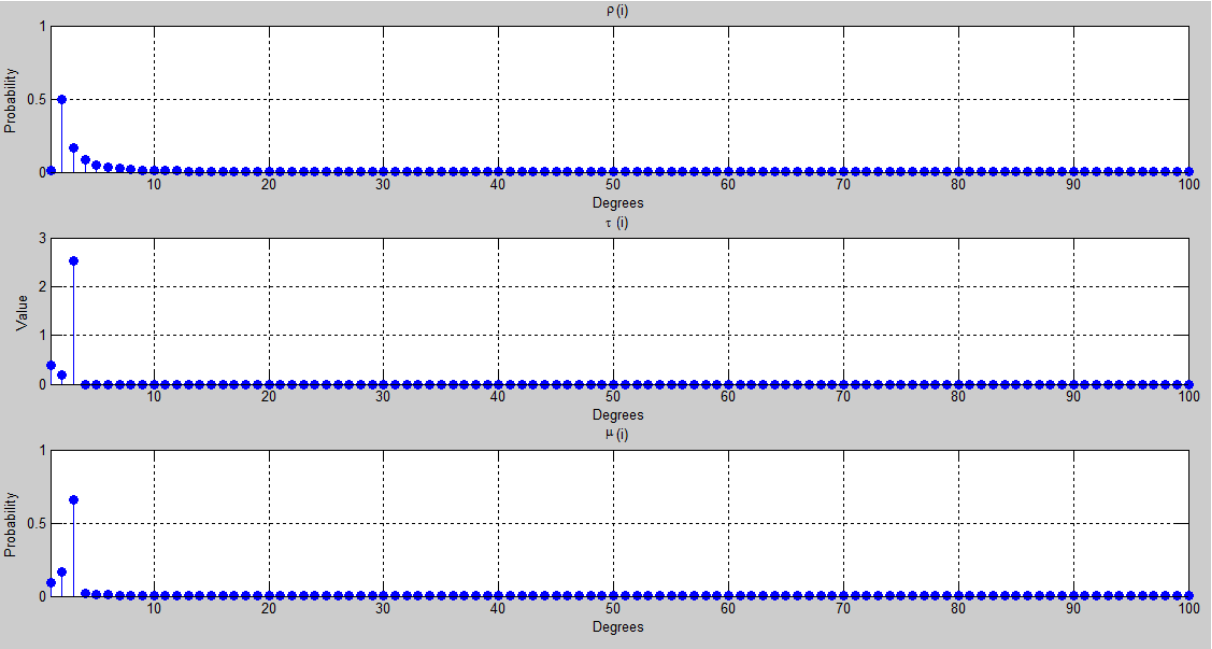


Figure D.1: Plots of $\rho(i)$, $\tau(i)$ and $\mu(i)$ for $k = 100,\ c = 0.5$ and $\delta = 0.05$

2) For $k = 100,\ c = 0.1$ and $\delta = 0.05$, $\mu(i)$ has two spikes at $i = 2$ (because of $\rho(i)$) and $i = \frac{100}{0.1 \cdot \ln\left(\frac{100}{0.05}\right)\sqrt{100}} \cong 13$ (because of $\tau(i)$).

In this case the expected Ripple size is R=$0.1 \cdot \ln\left(\frac{100}{0.05}\right)\sqrt{100} = 7.6$. In Figure D.2, plots of $\rho(i)$, $\tau(i)$ and $\mu(i)$ are given, respectively.

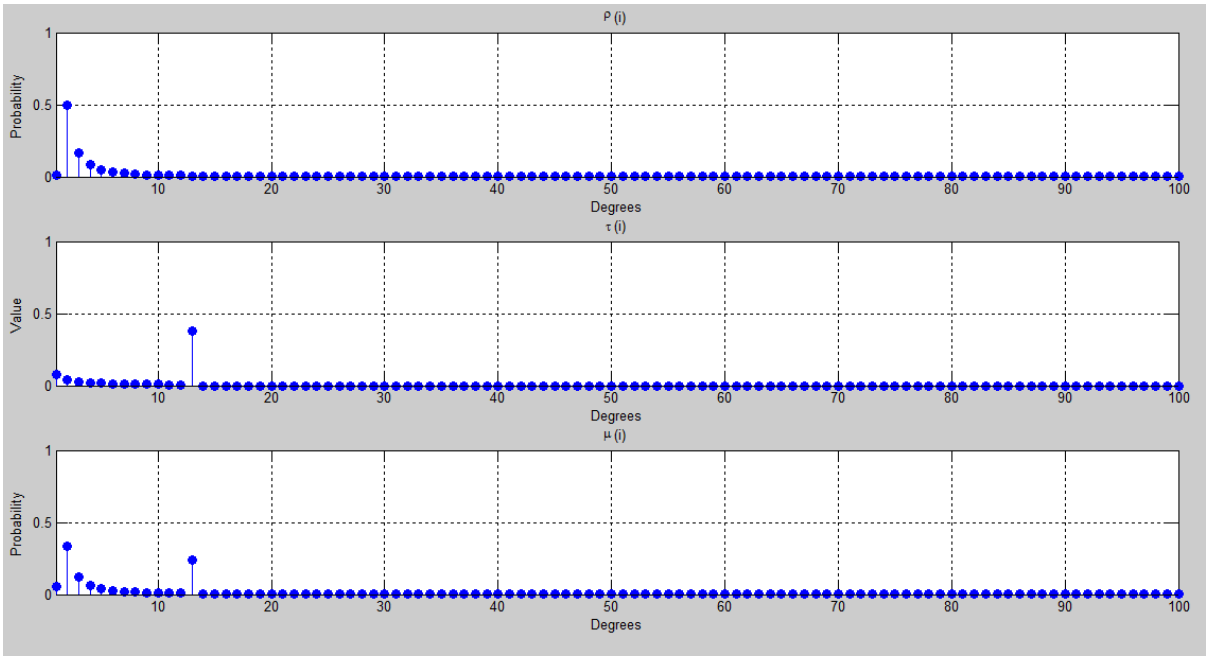


Figure D.2: Plots of $\rho(i)$, $\tau(i)$ and $\mu(i)$ for $k = 100$, $c = 0.1$ and $\delta = 0.05$

3) For $k = 100$, $c = 0.5$ and $\delta = 0.5$, $\mu(i)$ has two spikes at $i = 2$ (because of $\rho(i)$) and $i = \frac{100}{0.5 \cdot \ln\left(\frac{100}{0.5}\right)\sqrt{100}} \cong 4$ (because of $\tau(i)$).

In this case the expected Ripple size is R=$0.5 \cdot \ln\left(\frac{100}{0.5}\right)\sqrt{100} = 26.49$. In Figure D.3, plots of $\rho(i)$, $\tau(i)$ and $\mu(i)$ are given, respectively.

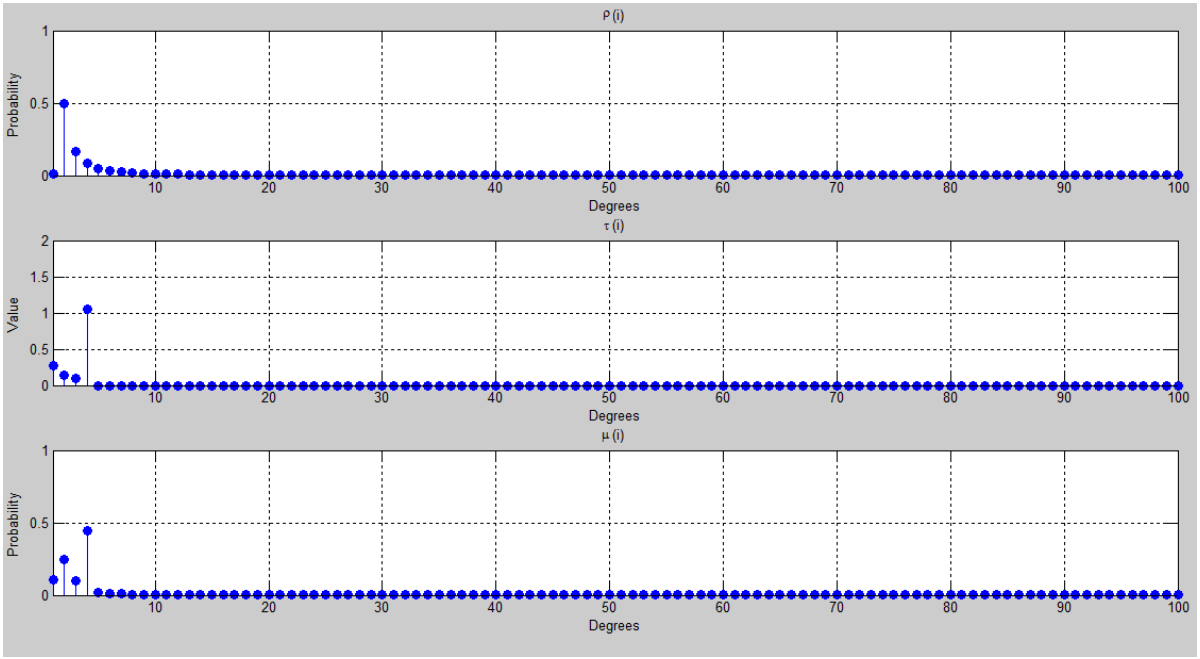


Figure D.3: Plots of $\rho(i)$, $\tau(i)$ and $\mu(i)$ for $k = 100$, $c = 0.5$ and $\delta = 0.5$

4) For $k = 100$, $c = 0.1$ and $\delta = 0.5$, $\mu(i)$ has two spikes at $i = 2$ (because of $\rho(i)$) and $i = \frac{100}{0.1 \cdot \ln\left(\frac{100}{0.5}\right)\sqrt{100}} \cong 19$ (because of $\tau(i)$).

In this case the expected Ripple size is R=$0.1 \cdot \ln\left(\frac{100}{0.5}\right)\sqrt{100} = 5.29$. In Figure D.4, plots of $\rho(i)$, $\tau(i)$ and $\mu(i)$ are given, respectively.

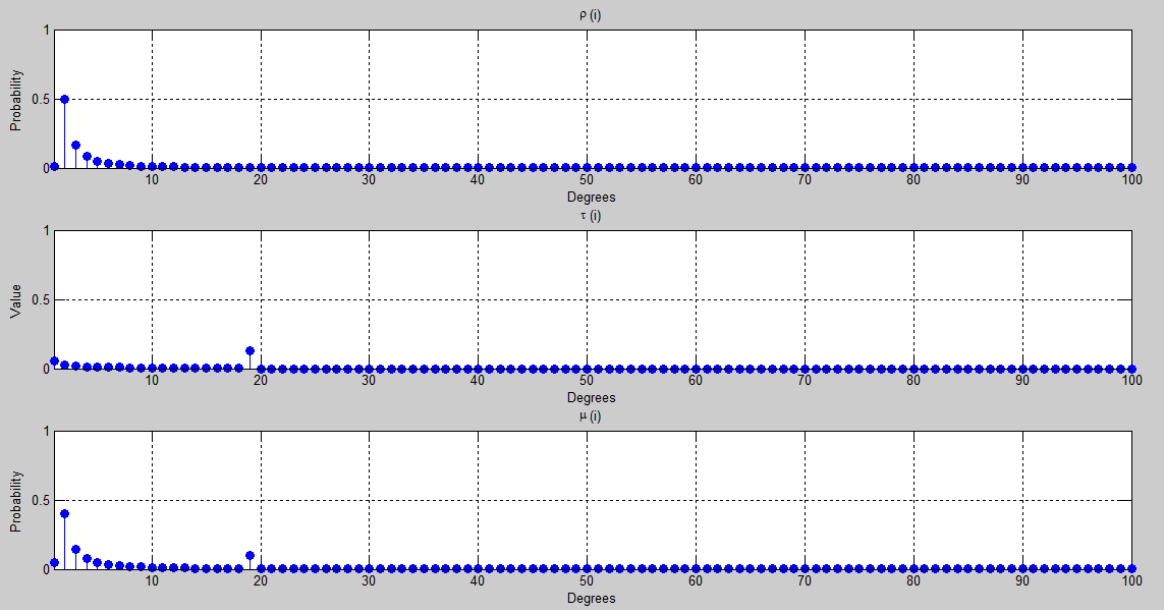


Figure D.4: Plots of $\rho(i)$, $\tau(i)$ and $\mu(i)$ for $k = 100$, $c = 0.1$ and $\delta = 0.5$

5) It was previously indicated that the values of c and δ must be chosen by taking the number of input symbols into account, otherwise the obtained degree distribution would not be reasonable. For instance, let us choose $k = 100$, $c = 0.01$ and $\delta = 0.5$. Then, $\mu(i)$ must have two spikes at $i = 2$ (because of $\rho(i)$) and $i = \frac{100}{0.01 \cdot \ln\left(\frac{100}{0.5}\right)\sqrt{100}} \cong 189$ (because of $\tau(i)$). Actually, this cannot be assessed as a spike since $\rho(i)$ and $\mu(i)$ are defined for $1 \leq i \leq k$. Even if $\tau(189)$ were defined, its extremely small value, $\tau(189)$=0.0003, would not effect the distribution at all. Furthermore, the expected Ripple size R=$0.01 \cdot \ln\left(\frac{100}{0.5}\right)\sqrt{100} = 0.53$, is too small and would probably cause the Ripple to disappear before the decoding ends. For all these reasons, this combination of c and δ is not proper when k=100. In Figure D.5, plots of $\rho(i)$, $\tau(i)$ and $\mu(i)$ are given, respectively and it is observed that the Robust

Soliton distribution $\mu(i)$ corresponding to these parameters is effectively the same as the Ideal Soliton distribution $\rho(i)$.

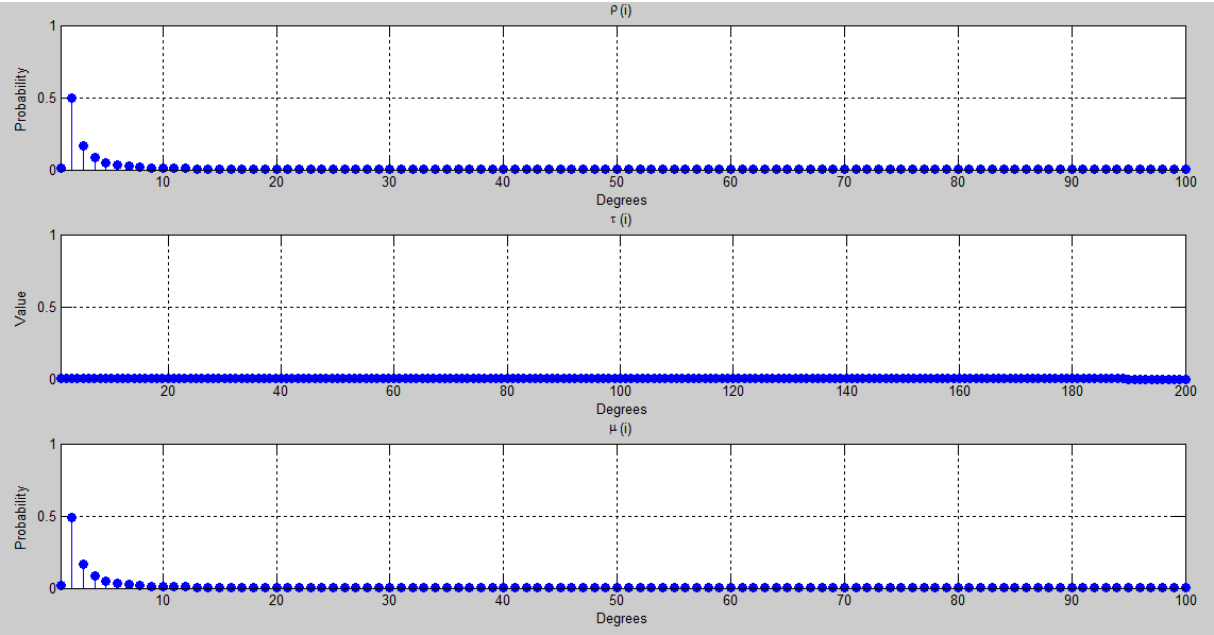


Figure D.5: Plots of $\rho(i)$, $\tau(i)$ and $\mu(i)$ for $k = 100$, $c = 0.01$ and $\delta = 0.5$

Throughput improvement with decreasing c can be explained by comparing the expected Ripple sizes of the first (R=38) and the second (R=7.6), which is much more reasonable than the first; and similarly, the third (R=26.49) and the fourth (R=5.29) cases. On the other hand, throughput improvement with increasing δ can be somehow understood by comparing the expected Ripple sizes of the first (R=38) and the third (R=26.49), and the second (R=7.6) and the fourth (R=5.29) cases.