2013

Design Of Fountain Codes With Error Control

Amrit Kharel
University of Mississippi

Design of Fountain Codes with Error Control

A Thesis
presented for the
Master of Science Degree in Engineering
Science with Emphasis in Telecommunications
The University of Mississippi

Amrit Kharel

June 2013

Abstract

This thesis is focused on providing unequal error protection (UEP) to two disjoint sources which are communicating to a common destination via a common relay by using distributed LT codes over a binary erasure channel (BEC), and designing fountain codes with error control property by integrating LT codes with turbo codes over a binary input additive white Gaussian noise (BI-AWGN) channel.

A simple yet efficient technique for decomposing the RSD into two entirely different degree distributions is developed and presented in this thesis. These two distributions are used to encode data symbols at the sources and the encoded symbols from the sources are selectively XORed at the relay based on a suitable relay operation before the combined codeword is transmitted to the destination. By doing so, it is shown that the UEP can be provided to these sources.

The performance of LT codes over the AWGN channel is well studied and presented in this thesis which indicates that these codes have weak error correction ability over the channel. But, errors introduced into individual symbols during the transmission of information over noisy channels need correction by some error correcting codes. Since it is found that LT codes alone are weak at correcting those errors, LT codes are integrated with turbo codes which are good error correcting codes. Therefore, the source data (symbols) are at first turbo encoded and then LT encoded and transmitted over the AWGN channel. When the corrupted encoded symbols are received at receiver, LT decoding is conducted followed by turbo decoding. The overall performance of the integrated system is studied and presented in this thesis, which suggests that the errors left after LT decoding can be corrected to some extent by turbo decoder.

Acknowledgments

First and foremost, I would like to express my sincere gratitude to my advisor, Dr. Lei Cao, for the continuous support, guidance and encouragement during the course of this work. I am especially grateful for his insightful suggestions and inputs over numerous discussions, which helped me understand the research problems clearly and carrying out the further research with much enthusiasm.

My heartfelt thank goes to Prof. Dr. John N. Daigle for the opportunity that he provided to me to start my Masters here at OleMiss. I would also like to thank him as well as Dr. Ramanarayanan "Vish" Viswanathan for consenting to be a member on the committee for my Master's defense. I thank my friend Hussein Fadhel for his valuable advice and friendly help. He answered a lot of my questions with great enthusiasm during his stay at OleMiss.

Lastly, I would like to thank my family. Their ever-green love and continuous support have helped me come to this point of my life. I can't describe this in words. I thank my wife Rojina for her all the support and love.

Table of Contents

List of Figures

Chapter 1

Introduction

The main objective of this thesis is to integrate Luby Transform (LT) codes and turbo codes for transmission of information over noisy channels so that better overall performance of the system can be achieved over the performance of LT codes alone. Although LT codes are good erasure correcting codes, they perform poorly over noisy channels, such as additive white Gaussian noise (AWGN) channel. The idea is to determine whether or not turbo pre-coding is effective in eliminating errors left over after LT decoding.

In this thesis, we have also worked on providing unequal error protection (UEP) to two disjoint sources which are communicating to a common destination via a common relay. To achieve this goal, we have used two entirely different but appropriately designed degree distributions to encode the source data at the sources and adopted a suitable relay operation to selectively XOR the encoded symbols from the sources before transmitting them to destination.

1.1 Problems and Thesis Work

Fountain codes are record-breaking sparse-graph codes for channels with erasures, such as the Internet. In erasure channels, either the encoded packets are received without error or simply not received, that is, the packets with errors are discarded. Once the enough number of encoded error-free packets is received at the receiver, the original information can be decoded by using suitable decoding algorithm. But, over noisy channels such as AWGN channel, errors will be introduced into individual bits, that is, each of the transmitted bits

is corrupted by noise. As fountain codes are erasure correcting codes, they are weak at correcting the errors occurring in bits introduced during the transmission of data. But, by using message passing algorithm (MPA) and propagating the soft information along the tanner graph over a number of iterations, the fountain decoder can provide some useful information about the transmitted bits. Since turbo codes are known as a class of limit-approaching error correcting codes, we can argue that by combining these two codes together, it is possible that the turbo decoder may exploit the useful soft information obtained from the fountain decoder and correct the errors that are remained after fountain decoding. Therefore, in this thesis, we have integrated LT codes with rate compatible punctured turbo (RCPT) codes and studied their performances over AWGN channel. To do this, the information symbols at the sources are at first encoded using turbo encoder and then the turbo encoded symbols are LT encoded which are then transmitted over the channel. The transmitted symbols are corrupted over the noisy channel and these corrupted symbols, which we call soft information, received at the receiver are first fed into LT decoder which employs message passing algorithm to produce log likelihood ratio (LLR) values of the transmitted bits. These LLRs are further processed by turbo decoder using iterative decoding algorithm yielding final soft outputs (LLRs) based on which final decision about the transmitted bits is made. The simulations results are also presented which clearly show that the errors left after LT decoding can be further corrected by turbo decoder.

[1] discusses two sources communicating to a sink via a common relay where both the sources use the same degree distribution (deconvolved soliton distribution) for encoding the data symbols. The encoded symbols are selectively XORed at the relay and are transmitted to the sink such that the degree of the received symbols follows the RSD. Since the same degree distribution is used, each of the sources are equally protected. But, when the information from one of the sources is more important than the information from the other source, then the information from the more important source must be more protected than

the other. Therefore, if the RSD can be decomposed into two different distributions (say $p_1(.)$ and $p_2(.)$) and these two distributions are used to encode data at two sources, then one of the degree distributions may be able to provide better protection of data at one source than the other, that is, the sources can be unequally protected. To implement this idea, a simple yet efficient technique for decomposing the RSD into two entirely different distributions by deconvolution is developed and implemented in this thesis. The data at the sources are encoded by using these two different distributions and are transmitted along the relay where the encoded symbols from the two sources are selectively XORed using a suitable relay operation making sure that the degree of the received symbols at the receiver follows the RSD, which is so far the best degree distribution guaranteeing a good decoding performance. It is shown that by using these degree distributions and employing a proper relay operation, it is possible to provide UEP to the sources. The simulation results are also presented to support the idea.

1.2 Organization of this thesis

The contents of this thesis are organized as follows. Chapter 2 provides a more detailed background on LT and distributed LT codes and discusses their properties, encoding and decoding procedures. The technique that is capable of providing unequal error protection to each of the two sources which are communicating to a common sink via a common relay will be presented along with some simulation results. In Chapter 3, iterative decoding techniques employed in low density parity check (LDPC) code will be discussed which will be further used for the decoding of LT codes over AWGN channel. The simulation results for the performance of LT codes over AWGN channel will also be presented. In chapter 4, we will provide background on convolutional and turbo codes along with the puncturing and interleaving concept. We will also discuss the turbo decoder based on maximum-a-posteriori (MAP) and iterative decoding algorithms. In Chapter 5, we will combine the punctured turbo codes with the LT codes so that two layer of encoding (turbo encoding

followed by LT encoding) will be conducted at the source side and two layer of decoding (LT decoding followed by turbo decoding) will be conducted at the receiver and finally the performance will be evaluated based on SNR vs BER. Some simulations results will also be presented. In Chapter 6, we will discuss future work and draw conclusions.

Chapter 2

Distributed LT Codes

2.1 Digital Fountain Codes and LT Codes

Digital fountain codes are record-breaking sparse-graph codes for channels with erasures, such as the Internet. Channels with erasures are of great importance. For example, files are chopped into packets and are transmitted over the Internet. These transmitted packets are either received without error or not received at the receiver, that is, packets with errors are erased. Common methods for communicating over such channels employ a feedback channel from receiver to sender that is used to control the retransmission of erased packets. For example, the receiver might send back messages that identify the missing packets, which are then retransmitted. Alternatively, the receiver might send back messages that acknowledge each received packet; the sender keeps track of which packets have been acknowledged and retransmits the others until all packets have been acknowledged. The advantage of such simple retransmission protocols is that they work regardless of erasure probability p. However, the drawback is the need of feedback channel. According to Shannon, there is no need for the feedback channel: the capacity of the forward channel is $(1-p)l$ bits, whether or not we have feedback. In the case of a broadcast channel with erasures, one sender broadcasts to many receivers, and each receiver receives a random fraction $(1-p)$ of the packets. If every packet that is missed by one or more receivers has to be retransmitted, those retransmissions will be terribly redundant since every receiver will have already received most of the retransmitted packets.

Therefore, the erasure-correcting codes that require no feedback or almost no feedback is of great importance. The classic block codes for erasure correction are called Reed-Solomon codes [2]. An (n, k) Reed-Solomon code has the ideal property that if any k of the n transmitted symbols are received then the original k source symbols can be recovered. However, Reed-Solomon codes have the disadvantage that they are practical only for small k and n. These codes lose efficiency for large k and n, requiring quadratic encoding/decoding time [3] [4]. Furthermore, with a Reed-Solomon code, as with any block code, one must estimate the erasure probability p and choose the code rate $r = k/n$ before transmission, that is, the rate is fixed. In the case when p is larger than expected and the receiver receives fewer than k symbols, decoding fails. To combat this problem, a better way is pioneered by Michael Luby at his company Digital Fountain.

The idea of digital fountain codes is as follows: The encoder is a fountain that produces an endless supply of water drops (encoded packets). Lets say the original source file has a size of kl bits, and each drop contains l encoded bits. Now, anyone who wishes to receive the encoded file holds a bucket under the fountain and collects drops until the number of drops in the bucket is a little larger than k. They can then recover the original file. Digital fountain codes are rateless in the sense that the number of encoded packets that can be generated from the source message is potentially limitless, and the number of encoded packets generated can be determined on the fly. Regardless of the statistics of the erasure events on the channel, we can send as many encoded packets as needed in order of the decoder to recover the source data. The source data can be decoded from any set of k' encoded packets, for k' slightly larger than k. Fountain codes also have fantastically small encoding and decoding complexity. They are called universal because they are simultaneously near optimal for every erasure channel.

Luby Transform (LT) codes, developed in [5], were invented by Michael Luby as random rateless codes for erasure channels. They are the first practical realization of fountain codes and are application layer codes. The symbol length for the codes can be arbitrary, from

one-bit binary symbols to general l-bit symbols. An LT code produces for a given set of k input symbols $x_1, x_2, \cdots, x_k$ a potentially limitless stream of output symbols $y_1, y_2, \cdots$ on the fly. The original k input symbols can be recovered from any set of n ($n > k$) output symbols with high probability. Therefore, these codes are called rateless codes. Encoding symbols can be generated as needed and sent over the erasure channel until a sufficient number have arrived at the decoder in order to recover the data. Therefore, rateless erasure codes have the potential of replacing transmission control protocol (TCP) which is based on automatic repeat request (ARQ)[6]. LT codes are usually considered in erasure channels. The distribution used for generating the output symbols lies at the heart of LT codes. They have sparse generator matrices, that is, the number of data symbols contributing to a code symbol is relatively small compared to the total number of data symbols and all the encoding and decoding operations are merely bit-wise XORs. Therefore, the encoding and decoding complexity is reduced.

LT codes are an excellent solution in a wide variety of situations. One is for storage: for example, to make a backup of a large file. We know that the magnetic tapes and hard drivers are all unreliable in the sense that some stored packets are permanently lost within one device over a period of time. Therefore, LT codes can be used to spray encoded packets all over the place, on every storage device available. Then to recover the backup file, one simply needs to find k' (slightly greater than k) packets from anywhere. Corrupted packets do not matter; we simply skip over them and find more packets elsewhere. Another application of LT codes is in broadcasting. For example, a broadcaster is broadcasting a movie to thousands of it subscribers. In a standard approach in which the file is transmitted as a plain sequence of packets with no encoding, each subscriber would have to notify the broadcaster of its missing packets, and request retransmission. And, with the thousands of subscribers all requesting such retransmissions, there would be retransmission request for almost every packet. However, if the broadcaster uses LT codes to encode the movie, each

subscriber can recover the movie from any k' (slightly larger than k) packets. This saves a lot of bandwidth and time.

2.1.1 Erasure Channel

Erasure channels constitute an important class of channels in communication. They are so important because many lossy or noisy communication channels can be modeled or simplified to approximate an erasure channel. Elias introduced the concept of erasure channels in 1955 [7]. The binary erasure channel shown in Figure 2.1 transmits one of two symbols, usually the binary digits $x \in \{0, 1\}$. However, the receiver either receives the bit correctly or it receives a message "e" that the bit was not received (it was erased). The BEC erases a bit with probability ε, called the erasure probability of the channel. Thus the channel transition probabilities for the BEC are:

$$\begin{aligned} &p(y = 0|x = 0) = 1 - \varepsilon, \quad P(y = e|x = 0) = \varepsilon, \quad p(y = 1|x = 0) = 0, \\ &p(y = 0|x = 1) = 0, \quad P(y = e|x = 1) = \varepsilon, \quad p(y = 1|x = 1) = 1 - \varepsilon. \end{aligned} \tag{2.1}$$

The BEC does not flip bits, so if y is received as a 1 or 0 the receiver can be completely certain of the value of x:

$$\begin{aligned} &p(x = 0|y = 0) = 1, \qquad p(x = 1|y = 0) = 0, \\ &p(x = 0|y = 1) = 0, \qquad p(x = 1|y = 1) = 1. \end{aligned} \tag{2.2}$$

The erasure channel is said to be memoryless, if the output of the channel at a given time is affected only by the current input at that time, and is independent of the other (past of future) inputs. Practical examples of channels that can be modeled as memoryless erasure channels include communication networks (e.g, the Internet and data storage devices).

An erasure channel can be a bit erasure channel or a packet erasure channel. A chosen erasure code can be on bit level or packet level accordingly. The communication between a

source and a destination occurs via packets in the case of the Internet. The data intended for the destination is partitioned into packets (binary strings) at the source and are then transmitted in succession to the destination over the network. Packets may be lost over the Internet or packets may be corrupted. The errors in the corrupted packets are detected by means of a cyclic redundancy check (CRC) and are discarded if errors occurred. The receiver only accepts errorfree packets. Thus, packets are either transmitted correctly or "erased" by the channel - in this case, the Internet.

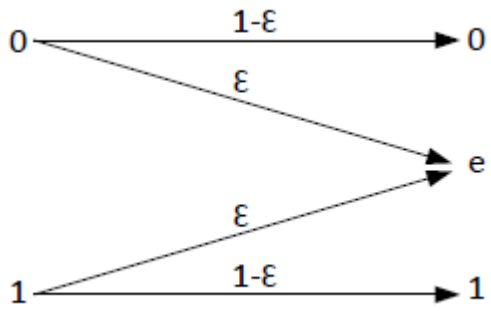


Figure 2.1. The binary erasure channel (BEC)

2.1.2 Encoding Procedure

For a given set of k data symbols, an encoding (code) symbol is generated by bitwise XORing of a randomly chosen subset of the data. The process of generating an encoding symbol is conceptually as follows:

- An integer d between 1 and k, called the degree of the code symbol, is randomly chosen for each code symbol from a degree distribution.
- A set of d data or information symbols is chosen uniformly at random from the set of k symbols. These d data symbols are called the neighbors of the code symbol to be generated.

- The d data symbols or neighbors are bitwise XOR-ed to produce the code symbol of degree d.

By this process, an arbitrary number of code symbols can be generated. Each code symbol is generated independently of all others.

2.1.3 Decoding Procedure

Bipartite graph is used for the decoding of LT codes where the data (source) and code symbols can be viewed as the vertices. Edges connect every code symbol to all its neighbors. Generator matrix is used to develop such a graph. When using the code symbols to recover the original input symbols of the data, the decoder needs to know the degree and set of neighbors of each code symbols. Therefore, the knowledge of the code graph is necessary for decoding an LT code.

Let n be the total number of code symbols generated from the data symbols (k) where $n > k$. The graph is shown in Figure 2.2. The belief propagation (BP) algorithm is used to

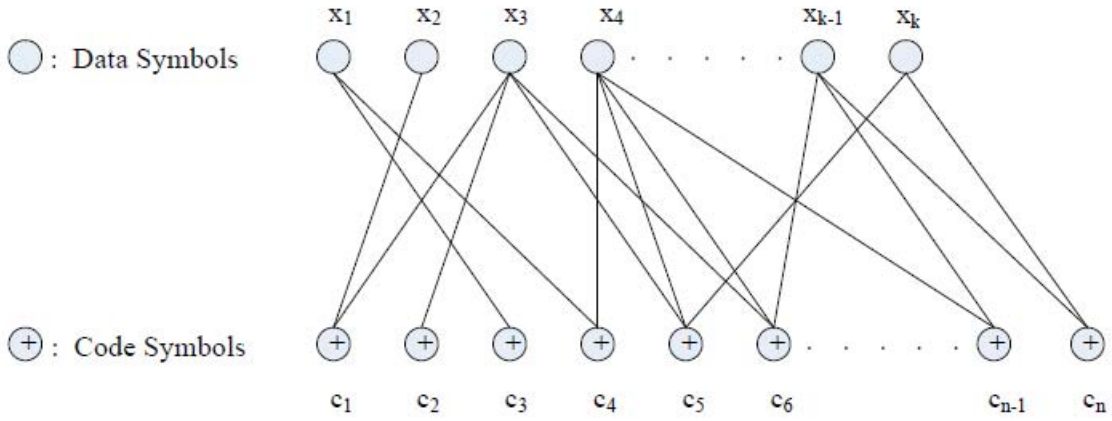


Figure 2.2. Graphical Representation of LT codes

decode the original data symbols. The decoding is done in the following manner:

1. The decoder identifies all code symbols of degree one (that is, those connected to a single source symbol) in the code graph. If there is at least one code symbols that has

exactly one neighbor, then the neighbor is recovered immediately since it is a copy of the code symbol. If there exist no degree one code symbols, the decoding fails.

2. The code symbols of degree one and the associated edges are erased from the graph.

3. The value of the recovered input symbol is XORed with any remaining code symbols that also have that input symbol as a neighbor. The corresponding edges are erased from the graph thus decreasing the degree of such code symbols by one.

4. The decoder repeats steps $1 - 3$ using the new reduced code graph.

If the decoder fails to recover all the k data symbols from n code symbols, a decoding failure occurs which can happen due to either of the following reasons:

1. If there exist information symbols that are not connected to any code symbol in the code graph.

2. If the decoding fails before all the data has been decoded due to the absence of degree one code symbols.

Therefore, the degrees and the neighbors of the code symbols chosen randomly during encoding procedure are very important to avoid or minimize the decoding failure. For this, degree distribution plays the vital role.

2.1.4 The Robust Soliton Distribution (RSD)

The most important part of LT codes is the degree distribution which gives the probability distribution of degrees of the code symbols. LT codes use a specially constructed degree distribution called the robust soliton distribution [5]. The RSD ensures that the average number of degree one code symbols is large enough at each point in the decoding process so that it never disappears completely with high probability.

The robust soliton distribution is represented by $\mu(\cdot)$. For constants $c > 0$ and $\delta \in (0, 1]$, the probability mass function $\mu(\cdot)$ is given by

$$\mu(i) = \frac{\rho(i) + \tau(i)}{\beta}, \quad \text{for } 1 \leq i \leq k, \tag{2.3}$$

where

$$\beta = \sum_{i=1}^{k} (\rho(i) + \tau(i)) \tag{2.4}$$

The $\rho(i)$ (a probability mass function on $1 \leq i \leq k$) and $\tau(i)$ are given by

$$\rho(i) = \begin{cases} \frac{1}{k}, & \text{for} \quad i = 1, \\ \frac{1}{i(i-1)}, & \text{for} \quad 2 \leq i \leq k, \end{cases} \tag{2.5}$$

$$\tau(i) = \begin{cases} \frac{S}{ik}, & \text{for} \quad 1 \leq i \leq \frac{k}{S} - 1, \\ \frac{S \ln(\frac{S}{\delta})}{k}, & \text{for} \quad i = \frac{k}{S}, \\ 0, & \text{otherwise.} \end{cases} \tag{2.6}$$

The parameter S represents the average number of degree one code symbols and is defined as

$$S = c \cdot \sqrt{k} \cdot \ln\left(\frac{k}{\delta}\right). \tag{2.7}$$

Luby has suggested in [5] that the k data symbols can be recovered from any set of n code symbols with probability at least $1 - \delta$ and n is given by

$$\begin{aligned} n &= k\beta \\ &= k + c \cdot \sqrt{k} \cdot \ln^2(k/\delta). \end{aligned} \tag{2.8}$$

As shown above, the RSD has two components: $\rho(\cdot)$ and $\tau(\cdot)$. The support of $\rho(i)$ is extending over the entire range of degrees from 1 through k while $\tau(i)$ is restricted to i in

the range 1 through k/S. The distribution $\mu(i)$ has the spikes at two points: $i = 2$ and $i = k/S$. The spike in $\tau(i)$ at $i = k/S$ is to ensure that each of the data symbols is likely to be connected to at least one code symbol. The plots for $\rho(\cdot)$, $\tau(\cdot)$ and $\mu(\cdot)$ are shown in Figures 2.3, 2.4 and 2.5 respectively.

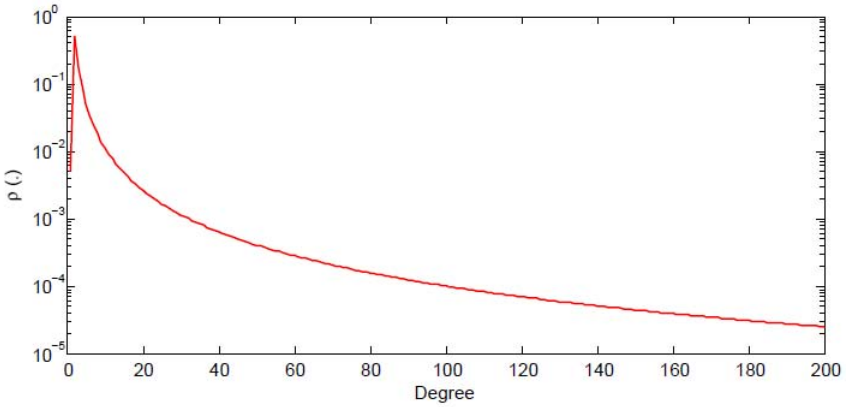


Figure 2.3. $\rho(\cdot)$ **component of RSD for** $k = 200, c = 0.05, \delta = 0.5$**.**

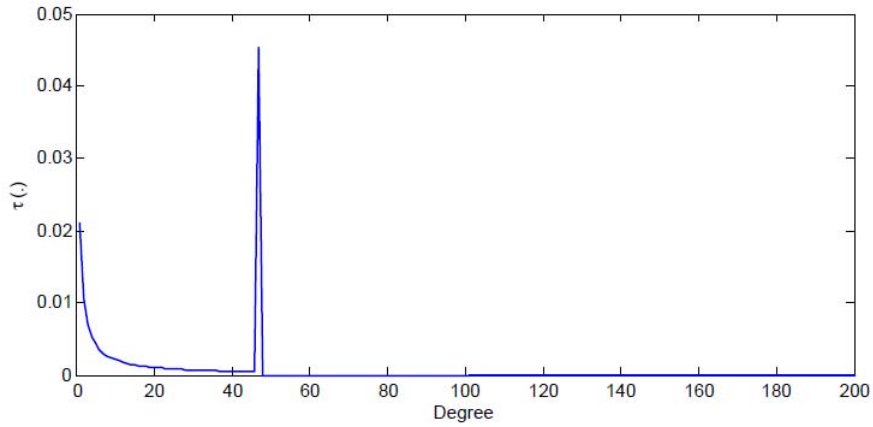


Figure 2.4. $\tau(\cdot)$ **component of RSD for** $k = 200, c = 0.05, \delta = 0.5$**.**

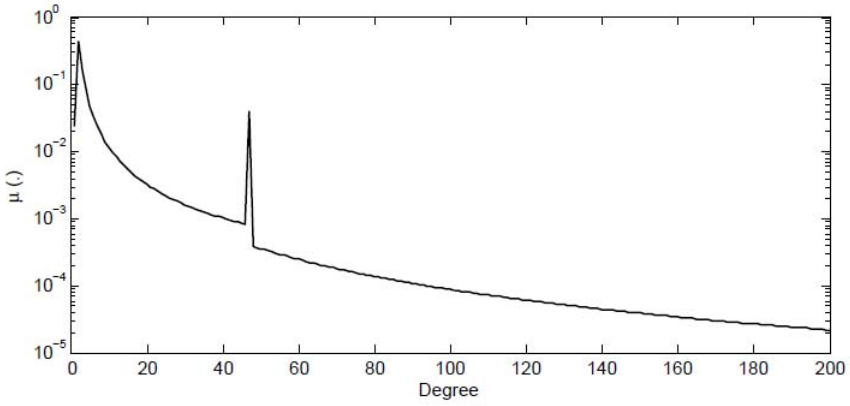


Figure 2.5. RSD $\mu(\cdot)$ for $k = 200, c = 0.05, \delta = 0.5$.

2.1.5 Overhead in LT codes

The extra number of code symbols that are required for decoding k data symbols with probability at least $1 - \delta$ is called the overhead which is given by

$$\begin{aligned} m &= n - k \\ &= c \cdot \sqrt{k} \cdot \ln^2(k/\delta). \end{aligned} \tag{2.9}$$

The fraction m/k is called the fractional overhead of LT codes and is given as $\frac{c \cdot \ln^2(k/\delta)}{\sqrt{k}}$. So, if $k \to \infty$, $m/k \to 0$. This means that LT codes are asymptotically optimal. On the other hand, as the number of data symbols k decreases, the fractional overhead increases which is observed in simulations as well.

2.2 Distributed LT codes

In [1] a novel distributed encoding procedure is proposed to realize codes that resemble LT codes in both structure and performance which are called the distributed LT Codes. For the case of two sources communicating with a single sink via a common relay, $k/2$ source symbols are separately encoded into slightly more than k code symbols at each source. These two codewords are then selectively XOR-ed at the relay, such that the result can be decoded

at receiver to recover all k information symbols. The primary advantage of this technique is that it exploits the improved efficiency of such codes for large block lengths; by forming a single codeword at the destination representing all the distributed data sets. Thus, there is a reduction in the overhead compared with a system in which each data set is encoded with its own LT code having a smaller block length.

As suggested in [1][8], consider a scenario in which two independent sources in a network transmit information to a sink through a common relay, that is, multiple access relay channel as shown in Figure 2.6.

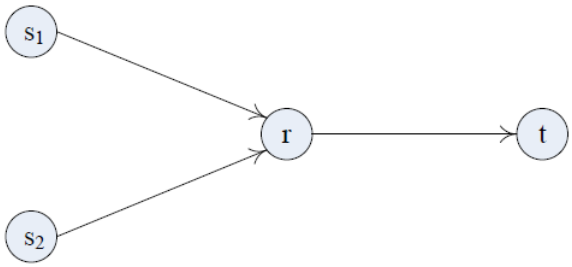


Figure 2.6. A two-source single-sink relay network.

The relay is assumed to have limited data processing capability, and communication between the sources is impossible or not desired. Also, the relay has limited memory - specifically, it can store only one packet (symbol) per source at any given time. In such a situation, there are two distinct LT-based approaches to protect the data against erasures as discussed below:

- Two different LT codes are used to independently encode the data at these two source and the resulting two encoded sequences are time-multiplexed through the relay.

- The data at these two sources are encoded in such a way that the relay combines its inputs in some low-complexity operation and transmits a combined sequence which "looks like" a single LT codeword.

The second scheme possesses the following advantages over the first:

- LT codes over larger information blocklengths (such as those in the second scheme) possess a smaller fractional reception overhead as discussed in Section 2.1.5.

- For fixed-rate coding techniques, a larger codeword length results in a smaller probability of decoding failure. For a given number of code symbols transmitted by the relay, the second scheme can deliver fewer, longer LT codewords to the sink than the first scheme.

The straightforward collective encoding of the sources' data at the relay places a heavy computational as well as memory requirement on the relay. The relay has to store all the data symbols from both the sources before it can start encoding. This also leads end-to-end transmission delay, as the relay initially needs to spend more time collecting data from the sources. Therefore, some encoding operations that can be carried out at the sources themselves help to minimize the computation at the relay as well as the delay in transmission. To achieve this, a novel scheme [8] is developed by the means of which a code resembling an LT code can be delivered to the sink. The basic steps involved in this scheme are as follows:

- The data symbols at each of the two sources are encoded by a specially designed code called a *distributed LT* (DLT) code.

- The DLT code symbols from the two sources are combined at relay into an LT-like code, called a *modified LT* (MLT) code. The combining operation at the relay consists of selectively XORing of the DLT code symbols coming from the two sources.

The MLT code is similar to an LT code in the sense that its degree distribution approximately follows the RSD. The two sources require some degree distributions based on which encoding can be done and DLT code symbols can be generated.

2.2.1 Decomposition of an LT code into two DLT codes

Consider the network as shown in Figure 2.6. Let the two sources s_1 and s_2 each have a block of $\frac{k}{2}$ data symbols required to transmit to the sink t, denoted by $\mathcal{D}_1$ and $\mathcal{D}_2$ respectively. Let X_1 be a code symbol generated at s_1 by XORing d_1 information symbols from $\mathcal{D}_1$, and X_2 be likewise generated at s_2 from $\mathcal{D}_2$ and is of degree d_2. Both d_1 and d_2 have the same degree distribution. Since X_1 and X_2 are combined, that is, selectively XORed at the relay, the code symbol being transmitted from the relay to the sink is $X_1 \oplus X_2$ and the corresponding degree is clearly $d_1 + d_2$. Here, the degrees of X_1 and X_2 should be chosen in such a way that the $d_1 + d_2$ should follow the RSD.

Assuming that d_1 and d_2 are chosen independently (that is, no co-operation among the sources) according to the same distribution $p(i)$, $1 \leq i \leq \frac{k}{2}$, $d_1 + d_2$ follows $p * p$, where "*" indicates convolution. Thus, to determine $p(\cdot)$ requires the deconvolution of the RSD. The RSD has support over $i = 1, 2, \cdots, k$. The distribution $p(\cdot)$ should have a support over $i = 1, 2, \cdots, k/2$ since it is to be used in encoding the data symbols at each source.

2.2.2 Deconvolution of RSD

Let $y(n)$ be a real sequence such that $y(n) = (x * x)(n) \quad \forall \quad n$. The process that takes $y(n)$ and produces $x(n)$ is called *deconvolution*. As suggested in [1], direct deconvolution of the RSD does not yield a valid degree distribution $p(\cdot)$. The problems specific to the RSD are:

1. For $k \geq 2$, the only way of ensuring that $(p * p)(1) = \mu(1) > 0$ is if we let $p(0) > 0$. This violates the condition on the support of $p(\cdot)$. Moreover, $p(0) > 0$ implies that $X_1 \oplus X_2$ could have degree zero, that is, $\mu(0) = 0$ which is not permitted by the RSD and is wasteful.

2. If the degree-one symbols in the RSD are completely disregarded and $\mu(i)$ is attempted

to be reproduced for $i > 1$ by recursively solving for $p(\cdot)$ from

$$\begin{aligned} p^2(1) &= \mu(2), \\ 2 \cdot p(1) \cdot p(2) &= \mu(3), \\ 2 \cdot p(3) \cdot p(1) + p^2(2) &= \mu(4), \\ 2 \cdot p(4) \cdot p(1) + 2 \cdot p(3) \cdot p(2) &= \mu(5), \\ &\vdots \end{aligned} \tag{2.10}$$

then a negative value for $p(k/S)$ will be obtained due to the "spiky" behavior of the RSD at k/S.

3. The support of $p(i)$ is $i = 1, \cdots, k/2$ since each source have only $k/2$ data symbols. When 2.10 is solved for $p(i)$, only the first $\frac{k}{2} + 1$ degrees of the RSD are made used. Consequently, the "tail" of the RSD (for $i = k/2 + 2, \cdots, k$) can not be reproduced by $p * p$.

4. Finally, if we restrict ourselves to deconvolving a portion of the RSD as dictated by the preceding constraint, then it is not necessarily true that the resulting $p(i)$ will sum to one.

To avoid the direct deconvolution, the RSD $\mu(\cdot)$ is first split into two distributions: $\mu'(\cdot)$ and $\mu''(\cdot)$ such that $\mu''(\cdot)$ captures the problematic part of the RSD (that is, the degree one symbols and the spike at $i = k/S$) and $\mu'(\cdot)$ is a smooth distribution that can be easily deconvolved.

Assume that $\rho(\cdot)$ and $\tau(\cdot)$ are given by 2.5 and 2.6, respectively. Then, $\mu'(\cdot)$ is defined as follows:

$$\mu'(i) = \begin{cases} 0, & \text{for } i = 1 \\ \frac{\rho(i)+\tau(i)}{\beta'}, & \text{for } 2 \leq i \leq \frac{k}{S} - 1 \\ \frac{\rho(i)}{\beta'}, & \text{for } \frac{k}{S} \leq i \leq k \end{cases} \tag{2.11}$$

with the normalization factor β' given by

$$\beta' = \sum_{i=2}^{k} \rho(i) + \sum_{i=2}^{k/S-1} \tau(i). \tag{2.12}$$

Similarly, $\mu''(\cdot)$ is defined as

$$\mu''(i) = \begin{cases} \frac{\rho(i)+\tau(i)}{\beta''}, & \text{for } i = 1 \\ \frac{\tau(\frac{k}{S})}{\beta''}, & \text{for } i = \frac{k}{S} \\ 0, & \text{otherwise} \end{cases} \tag{2.13}$$

with the normalization factor β'' given by

$$\beta'' = \rho(1) + \tau(1) + \tau\left(\frac{k}{S}\right) \tag{2.14}$$

Thus, noting that $\beta' + \beta'' = \beta$, from 2.4 the RSD can be rewritten as

$$\begin{aligned} \mu(i) &= \frac{\beta'}{\beta} \cdot \mu'(\cdot) + \frac{\beta''}{\beta} \cdot \mu''(i) \\ &= \frac{\beta'}{\beta} \cdot \mu'(\cdot) + \left(1 - \frac{\beta'}{\beta}\right) \cdot \mu''(i), \quad \text{for} \quad 1 \leq i \leq k. \end{aligned} \tag{2.15}$$

Hence, the RSD $\mu'(\cdot)$ can be viewed as a mixture of the distributions $\mu'(\cdot)$ and $\mu''(\cdot)$ with mixing parameter $\frac{\beta'}{\beta}$.

Now, $\mu'(\cdot)$ is deconvolved to yield a valid probability distribution $f(\cdot)$, and the final degree distribution $p(\cdot)$ is obtained as a mixture of $f(\cdot)$ and $\mu''(\cdot)$ which is used to encode the data symbols at the sources.

Using the approach in 2.10, the smooth distribution $\mu'(i)$ is deconvolved and the result is used to construct the DLT codes. Let's define the function $f(\cdot)$ as follow:

$$(f * f)(i) = \mu'(i) \quad \text{for} \quad 2 \leq i \leq k/2 + 1 \tag{2.16}$$

Then the solution to (2.16) is given by

$$f(i) = \begin{cases} \sqrt{\mu'(2)}, & \text{for } i = 1 \\ \frac{\mu'(i+1) - \sum_{j=2}^{i-1} f(j) f(i+1-j)}{2f(1)}, & \text{for } 2 \leq i \leq \frac{k}{2} \\ 0, & \text{otherwise} \end{cases} \tag{2.17}$$

The distribution $f(i) \geq 0$ for all i and $(f * f)(i) \approx \mu'(i)$ for $1 \leq i \leq k/2 + 1$ [1].

The final desired distribution $p(\cdot)$ which is obtained by mixing $f(\cdot)$ and $\mu''(\cdot)$ is called the *deconvolved soliton distribution (DSD)* and is given by

$$p(i) = \lambda \cdot f(i) + (1 - \lambda) \cdot \mu''(i), \quad \text{for} \quad 1 \leq i \leq k/2 \tag{2.18}$$

The mixing parameter λ is given by

$$\lambda = \sqrt{\frac{\beta'}{\beta}}. \tag{2.19}$$

2.2.3 Distributed Encoding

The deconvolved soliton distribution $p(\cdot)$ is used to encode data symbols in the network of Figure 2.6, such that the code symbols received by the sink t follow (approximately) the RSD in degree.

Initially, the DSD is used as the degree distribution at each of the two sources to encode the data symbols. The encoded symbols here is referred to as a DLT-2 code. The encoding procedure [1] is as follows:

1. The code symbol's degree d is generated by first randomly selecting either $f(\cdot)$ (with probability λ) or $\mu''(\cdot)$ (with probability $1-\lambda$) and then generating d with the selected distribution.

2. The code symbol's d neighbors are then selected equiprobably from among the $\binom{k/2}{d}$ possibilities, and are XORed to give the code symbol.

This process produces a sequence of DLT-2 code symbols. For every pair of DLT-2 code symbols (X_1, X_2) received from the two sources (s_1, s_2), the relay generates a code symbol Y which is a selective XOR of X_1 and X_2 in the following manner:

1. If both source symbols X_1 and X_2 were chosen according to the distribution $f(\cdot)$ component of the DSD, then $Y = X_1 \oplus X_2$. This occurs with probability $\lambda^2 = \frac{\beta'}{\beta}$.

2. If exactly one source symbol $X_i (i = 1 \quad \text{or} \quad 2)$ was chosen according to $\mu''(\cdot)$, then $Y = X_i$ and the other symbol is discarded.

3. If both X_1 and X_2 were chosen according to $\mu''(\cdot)$, then Y is randomly set to one of them with probability 0.5 and the other one is discarded.

The relay transmits the sequence of symbols Y to the sink. Consequently, the symbol transmitted to the sink is the XOR of two source symbols with probability $\lambda^2 = \frac{\beta'}{\beta}$ and has a degree distributed (approximately) according to $\mu'(\cdot)$, while the symbol transmitted to the sink is a copy of either X_1 or X_2 with probability $1 - \lambda^2 = 1 - \frac{\beta'}{\beta}$ and has a degree distributed according to $\mu''(\cdot)$. Thus, by 2.15, the degree of the symbol transmitted by the relay obeys (approximately) the RSD. In the operation described above, the relay must know from which distribution - $f(.)$ or $\mu''(.)$ - the degree of each encoded symbol it receives is drawn. This could be done by appending a single bit to the l-bit string making up each code symbol. Alternatively, if the relay knows the actual degree of each symbol it receives, it is possible to construct a randomized decision protocol [1] such that the relay transmits a symbol whose degree obeys (approximately) the RSD. The randomized decision rule is as follows:

1. Let X_1 and X_2 denote the symbols received from the two sources, and let d_1 and d_2 denote their degrees.

2. The relay generates two independent random variables U_1 and U_2, each uniformly distributed on $[0, 1]$.

3. The relay generates two binary random variables b_1 and b_2 as follows:

$$b_i = \begin{cases} 1, & \text{if} \quad \left(d_i = 1 \quad \text{and} \quad U_i < 1 - \frac{\lambda \cdot f(1)}{p(1)}\right) \\ & \text{or} \quad \left(d_i = 1 \quad \text{and} \quad U_i < 1 - \frac{\lambda \cdot f(k/S)}{p(k/S)}\right) \\ 0, & \text{otherwise.} \end{cases} \tag{2.20}$$

4. The relay then transmits the binary random variable Y defined as follows:

$$Y = \begin{cases} X_1 \oplus X_2, & \text{if} \quad b_1 = b_2 = 0 \\ X_1, & \text{if} \quad b_1 = 1 \quad \text{and} \quad b_2 = 0 \\ X_2, & \text{if} \quad b_1 = 0 \quad \text{and} \quad b_2 = 1 \\ \text{flip}(X_1, X_2), & \text{if} \quad b_1 = b_2 = 1 \end{cases} \tag{2.21}$$

Here, $\text{flip}(X_1, X_2)$ is a random variable taking the value of either X_1 or X_2 with equal probability.

The decoding procedure at the sink is exactly the same as LT decoding discussed in section 2.1.3.

2.3 Unequal Error Protection of the Sources

The initial studies on rateless codes considered the equal error protection (EEP) of all data. The EEP property would be sufficient for applications such as multicasting bulk data (e.g., a software file)[9]. However, in several applications, a portion of data may need more protection than the rest of data. For example, in an MPEG stream [10], I-frames need more protection than P-frames. Such applications raise a need for having codes with unequal error protection property. UEP codes were first studied in [11]. Rateless codes that can provide UEP property were for the first time discussed in [12]. Similarly, for the network

scenario of 2.6, information from one of the sources may be more important and needs better protection than that from the other source. So, this section is focused in developing a technique for providing unequal error protection to these sources.

2.3.1 Decomposing RSD into $p_1(\cdot)$ and $p_2(\cdot)$

Let $p_1(\cdot)$ and $p_2(\cdot)$ be the degree distribution used for encoding the data symbols at the sources s_1 and s_2 respectively. The code symbols coming from the sources s_1 and s_2 are selectively XORed at relay and are sent to the destination. We want the degree of the code symbols received at the sink to follow RSD.

We know from 2.15 that the RSD can be rewritten as:

$$\begin{aligned}\mu(i) &= \frac{\beta'}{\beta}\cdot\mu'(\cdot) + \frac{\beta''}{\beta}\cdot\mu''(i) \\ &= \frac{\beta'}{\beta}\cdot\mu'(\cdot) + \left(1-\frac{\beta'}{\beta}\right)\cdot\mu''(i), \quad \text{for} \quad 1\le i\le k.\end{aligned} \tag{2.22}$$

The distribution $\mu'(\cdot)$ is the smooth one. So, it can be deconvolved directly. Let the deconvolved components of $\mu'(\cdot)$ be $f_1(\cdot)$ and $f_2(\cdot)$. Then, we can write the following:

$$\begin{aligned} f_1(1)\cdot f_2(1) &= \mu'(2), \\ f_1(1)\cdot f_2(2) + f_1(2)\cdot f_2(1) &= \mu'(3), \\ f_1(1)\cdot f_2(3) + f_1(2)\cdot f_2(2) + f_1(3)\cdot f_2(1) &= \mu'(4), \\ f_1(1)\cdot f_2(4) + f_1(2)\cdot f_2(3) + f_1(3)\cdot f_2(2) + f_1(4)\cdot f_2(1) &= \mu'(5), \\ &\vdots \end{aligned} \tag{2.23}$$

If either of the distributions $f_1(\cdot)$ or $f_2(\cdot)$ is known, then the other can be calculated by using 2.23 easily. Let us suppose that the values of the distribution $f_1(\cdot)$ are known, then

we can write the following:

$$f_2(i) = \begin{cases} \frac{\mu'(2)}{f_1(1)}, & \text{for } i = 1 \\ \frac{\mu'(i+1) - \sum_{m=1}^{i-1} f_1(i+1-m) f_2(m)}{f_1(1)}, & \text{for } 2 \leq i \leq \frac{k}{2} \end{cases} \tag{2.24}$$

For 2.24 to give a valid distribution $f_2(\cdot)$, the other distribution $f_1(\cdot)$ should be similar in nature to $\mu'(\cdot)$, that is, $f_1(\cdot)$ must be decreasing with increasing values of i. Otherwise, convolution of $f_1(\cdot)$ and $f_2(\cdot)$ will not give $\mu'(\cdot)$ with the decreasing degree probabilities.

Since we know,

$$f_1(1) \cdot f_2(1) = \mu'(2), \tag{2.25}$$

we can set

$$f_1(1) = a \quad \text{where} \quad \mu'(2) \leq a \leq 1. \tag{2.26}$$

The value of a should never be set to any value that is less than $\mu'(2)$ because this makes the value $f_2(1) > 1$ which is undesirable. Now, we need to calculate all other values of the distribution, that is, $f_1(i)$ for $i \geq 2$. One of the simple ways to get these values is as follows:

$$f_1(i) = \frac{f(i)}{1 - f(1)} \cdot (1 - f_1(1)) \quad \text{for } 2 \leq i \leq \frac{k}{2} \tag{2.27}$$

where $f(\cdot)$ is the case when $f_1(\cdot)$ equals $f_2(\cdot)$, that is, when both the sources use the same distribution which is given in 2.17 and is as by

$$f(i) = \begin{cases} \sqrt{\mu'(2)}, & \text{for } i = 1 \\ \frac{\mu'(i+1) - \sum_{j=2}^{i-1} f(j) f(i+1-j)}{2 f(1)}, & \text{for } 2 \leq i \leq \frac{k}{2} \\ 0, & \text{otherwise} \end{cases} \tag{2.28}$$

Once we have $f_1(\cdot)$ and $f_2(\cdot)$, we can calculate the distributions $p_1(\cdot)$ and $p_2(\cdot)$ for the

sources s_1 and s_2 respectively as following:

$$p_1(i) = \lambda \cdot f_1(i) + (1 - \lambda) \cdot \mu''(i) \tag{2.29}$$

$$p_2(i) = \lambda \cdot f_2(i) + (1 - \lambda) \cdot \mu''(i) \tag{2.30}$$

where $\lambda = \sqrt{\frac{\beta'}{\beta' + \beta''}}$. The $\mu''(i)$, β' and β'' are given in 2.13, 2.12 and 2.14 respectively.

Now, we see that by adjusting a, we can obtain different sets of $p_1(\cdot)$ and $p_2(\cdot)$ which will be used to encode the data symbols at the sources s_1 and s_2 respectively. The degrees of the code symbols coming from each of the sources will vary from 1 to $k/2$. From 2.29 and 2.30, it is clear that a code symbol with degree d is generated by first randomly selecting $f_1(d)$ at source s_1 (similarly, $f_2(d)$ at source s_2) with probability λ, or $\mu''(d)$ with probability $1 - \lambda$.

2.3.2 Operation at Relay

Once the code symbols from the sources s_1 and s_2 with degrees according to the distributions p_1 and p_2 arrive at relay, they are selectively XORed and transmitted to the sink based on the following rules:

1. If the encoded symbols coming from the sources s_1 and s_2 are generated with the distributions $f_1(\cdot)$ and $f_2(\cdot)$ respectively, then they are XORed and the single combined symbol is transmitted to the sink.

2. If one of the encoded symbols coming from the two sources is encoded by using the distribution $\mu''(\cdot)$, then that particular symbol is transmitted to the sink while the other one is discarded.

3. If both the encoded symbols coming from the sources s_1 and s_2 are generated with the distribution $\mu''(\cdot)$, then one of these two symbols is randomly selected and is transmitted to the sink.

The detection of whether a coded symbol arriving at relay is generated by $f_i(\cdot)$ or $\mu''(\cdot)$ can be done by knowing the seeds of random generators in both the sources and the relay [5]. We see that the symbol transmitted to the sink is the XOR of two source symbols with probability $\lambda^2 = \beta'/\beta$ and the degree is distributed closely according to $\mu'(\cdot)$. Similarly, with the probability $1 - \lambda^2 = 1 - \beta'/\beta$, the degree of the symbol being transmitted to the sink is according to $\mu''(\cdot)$. Therefore, the degrees of the symbols being transmitted to the sink follow the RSD.

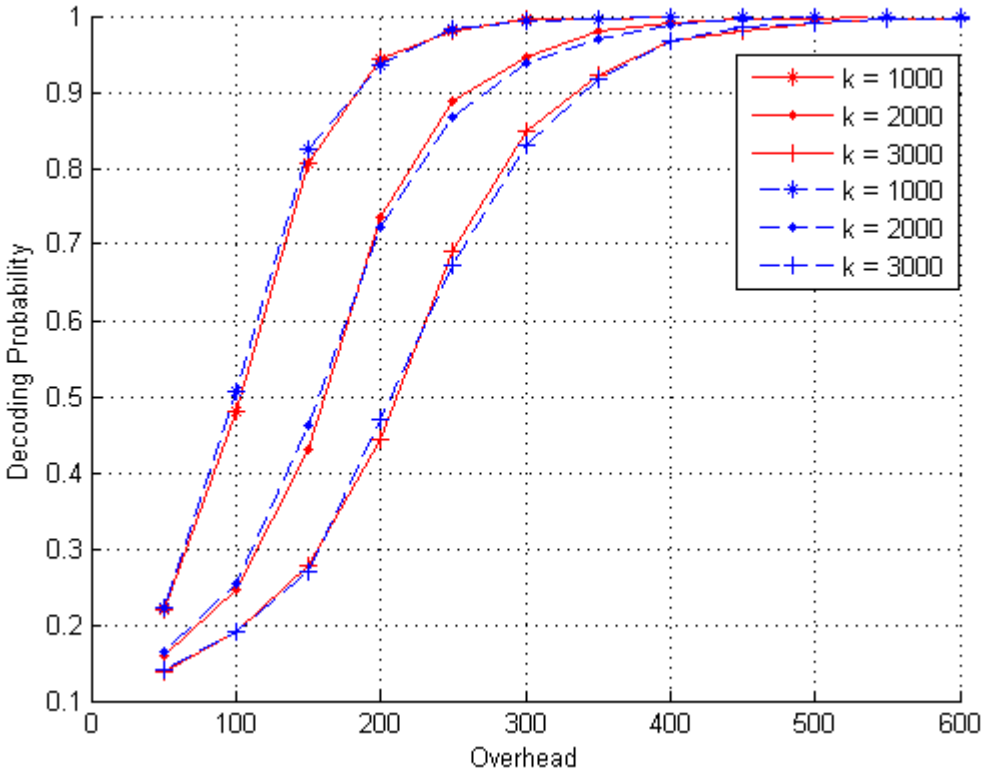


Figure 2.7. Comparison of average decoding probability. Dashed (or solid) lines represent the case when the sources use different distributions (or the same distribution). $f_1(1) = 0.80$ **and** (c,δ)**=(0.05, 0.5). Overhead is the number of encoded symbols additional to** k **that are also used in decoding.**

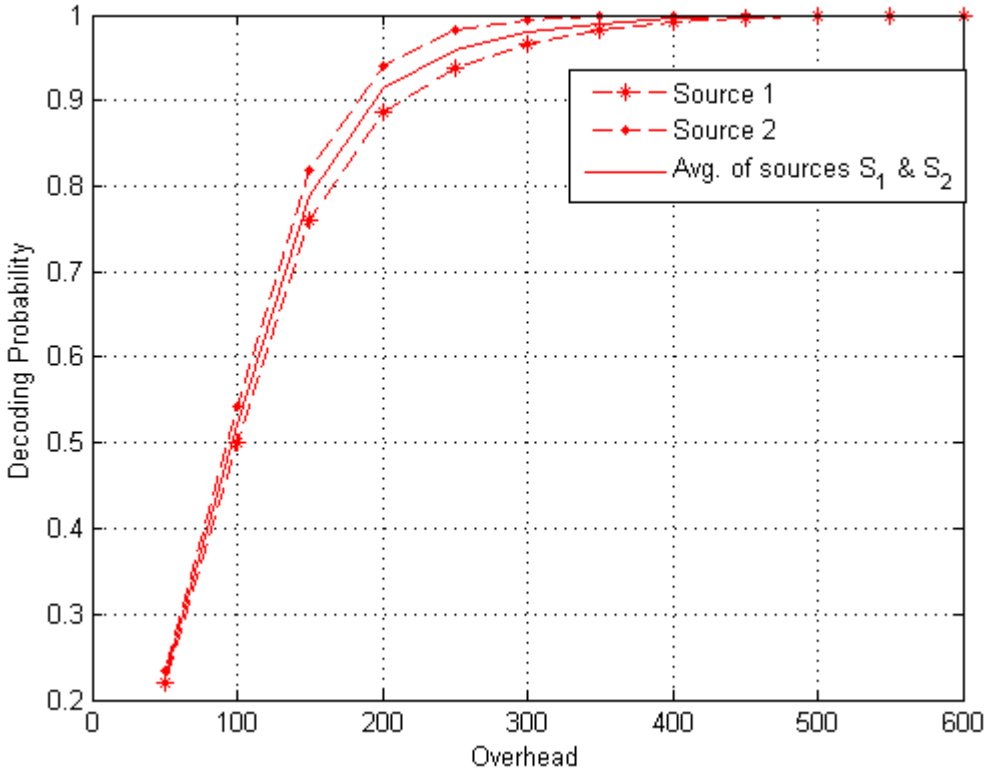


Figure 2.8. Decoding probability with different distributions, $k = 1000$ and $f_1(1) = 0.90$. UEP of s_1 and s_2 and the average decoding probability are shown. (c,δ)=$(0.05, 0.5)$.

2.3.3 Simulation and Results

Simulations were done in Matlab. Data symbols in the sources s_1 and s_2 were encoded using the distributions $p_1(\cdot)$ and $p_2(\cdot)$ respectively and were transmitted to the sink via relay based on the relay operation mentioned in 2.3.2.

The values shown in all the figures are the average values of the 2000 different values obtained from 2000 number of simulations. The overall decoding probabilities when the two sources used the different distributions are compared with that when both the sources used the same distribution which is depicted in Figure 2.7. We can see the comparable performances although using different distributions is slightly inferior to using the same distribution in terms of average decoding probability. However, by using different distributions at the two sources, we can observe the distinct difference between the decoding probabilities

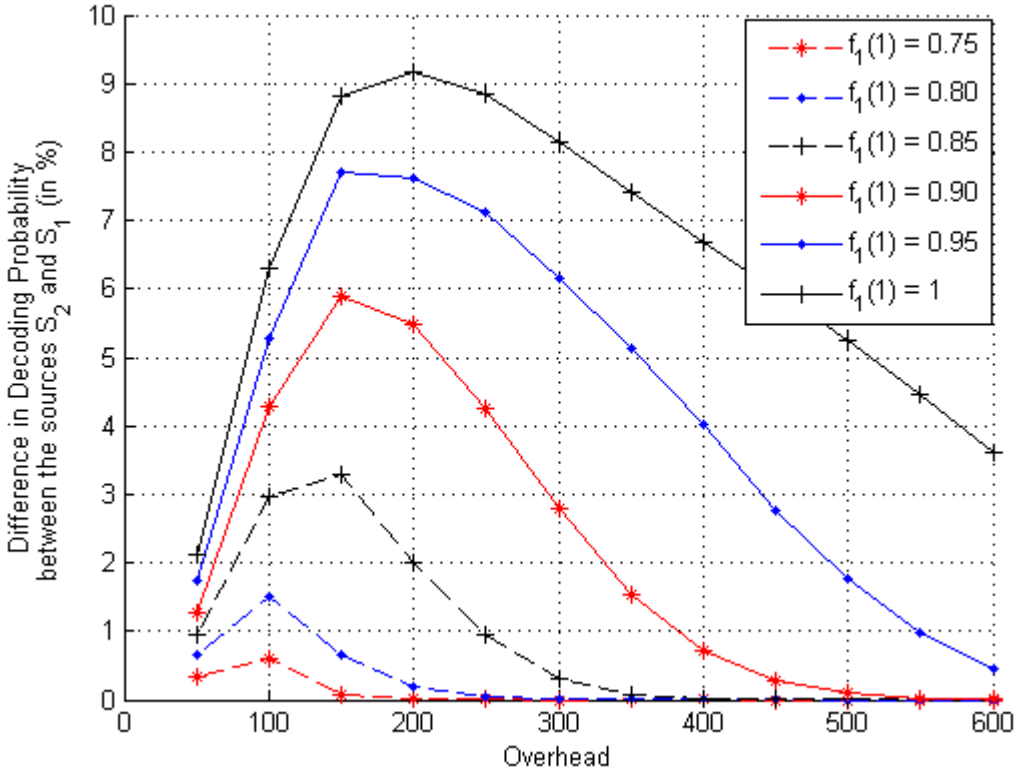


Figure 2.9. Difference in the decoding probabilities, (that is, decoding probability of s_2 minus that of s_1), when k = 1000 for different values of $f_1(1)$ and (c,δ)=(0.05, 0.5).

of the source s_1 and that of source s_2, as shown in Figure 2.8. Interestingly, it can also be observed that the source s_2 with a lower value of $f_2(1)$ is more protected and vice-versa. This is because a high $f_1(1)$ value leaves more symbols in s_1 uncovered at encoding and thus cannot be recovered at decoding. When one of the sources is more protected, the performance of the other is degraded to some extent. We observed that by varying the value a, the difference between the decoding probabilities of sources s_1 and s_2 changes. The difference between the decoding probabilities of two sources was calculated using $(ds_2 - ds_1)$,where, ds_2 is the decoding probability of s_2 and ds_1 is that of s_1, for different values of $f_1(1)$, that is, for $f_1(1) = 0.75$, $f_1(1) = 0.80$, $f_1(1) = 0.85$, $f_1(1) = 0.90$, $f_1(1) = 0.95$ and $f_1(1) = 1$ over the overhead range of $50 - 600$ at an interval of 50 and was plotted. As shown in Figure 2.9, the presented technique clearly suggests the unequal error protection for the two sources.

Chapter 3

LT Decoding Over BI-AWGN Channel

Originally, fountain codes were exclusively proposed for the reliable multicasting problem over the wired Internet, and therefore, have almost exclusively been investigated on erasure channels. The loss behavior on the wired Internet is appropriately modeled by an erasure channel as usually packets are dropped in intermediated routers. On BEC, theoretical analysis of the performance of fountain codes is feasible, and has been shown to have excellent performance. In fact, raptor codes [13] are currently being used by Qualcomm to provide fast and reliable transfer of large files over the Internet. Apart from erasure channels, there is a growing interest in exploiting rateless codes in physical layer. On channels such as binary symmetric channel (BSC) and additive white gaussian noise channel (AWGNC), there are also many potential applications (e.g., transfer of larger files over a wireless link, multicasting over a wireless channel). These issues have been addressed in [14] [15]. In this chapter we will present the decoding part of LT codes over AWGN channel and its performances. In noisy channel decoding, the message passing rules of fountain decoder resemble those of LDPC decoder. This chapter starts with an introduction to AWGN channel, LDPC codes and its decoding technique.

3.1 Binary Input AWGN channel

The noise sources on the communication channel can be man-made and natural types. The Gaussian noise channel we deal with is memoryless and the power spectral density

(PSD) of the noise is assumed to be the same for all frequencies. The PSD of AWGN is given by:

$$S_n(f) = \frac{N_0}{2}, \tag{3.1}$$

where N_0 is the single-sided power spectral density.

The binary-input AWGN channel can be described by the equation

$$y_i = x_i + z_i, \tag{3.2}$$

where $x_i \in \{-1, +1\}$ is the ith transmitted symbol, y_i is the ith received symbol and z_i is the additive noise sampled from a Gaussian random variable with mean 0 and variance σ^2. This is sometimes written as $z_i = \text{AWGN}(0, \sigma)$.

The probability density function for z is:

$$p(z) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-} \frac{z^2}{2\sigma^2}, \tag{3.3}$$

where $e^x = \exp(x)$ is the exponential function.

3.2 LDPC Codes

Low-density parity-check (LDPC) codes are a class of linear block codes which provide near capacity performance on a large collection of data transmission and storage channels while simultaneously admitting implementable decoders. LDPC codes were first proposed by Gallager in his 1960 doctoral dissertation [16]. The study of LDPC codes was resurrected almost after 35 years with the work of MacKay, Luby, and others [17] [18]. The matrix and graphical representation of the any codes is the most important part while applying message passing algorithms at the decoder.

3.2.1 Representation of LDPC Codes

The LDPC code is given by the null space of an $(n-k) \times n$ parity-check matrix H that has a low density of 1s where k is the number of data bits and n is the number of code bits. Tanner in 1981 [19] generalized LDPC codes and introduced a graphical representation of LDPC codes, now called Tanner graphs. The two types of nodes in a Tanner graph are the variable nodes (v-nodes) and the check nodes (c-nodes). The Tanner graph of a code is drawn according to the following rule: check node j is connected to variable node i whenever element h_{ji} in H is a 1. So, there are $n-k$ check nodes, one for each check equation, and n variable nodes, one for each code bit v_i. An example of H matrix for a $(10, 5)$ linear block code is shown below. The corresponding Tanner graph is shown in Figure 3.1. For all the check nodes, the sum of the neighboring positions among the variable nodes is zero, that is, the value of check node is always zero.

$$H = \begin{bmatrix} 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 1 \end{bmatrix} \tag{3.4}$$

3.2.2 Iterative Decoding Algorithm

In addition to introducing LDPC codes, Gallager also provided a decoding algorithm that is typically near optimal. Since that time, other researchers have independently discovered that algorithm and related algorithms. The algorithm iteratively computes the distributions of variables in graph-based models and comes under different names, depending on the context. These names include: the sum-product algorithm (SPA), the belief propagation algorithm (BPA), and the message passing algorithm (MPA). The term "message passing" usually refers to all such iterative algorithms, including the SPA, BPA and its approxima-

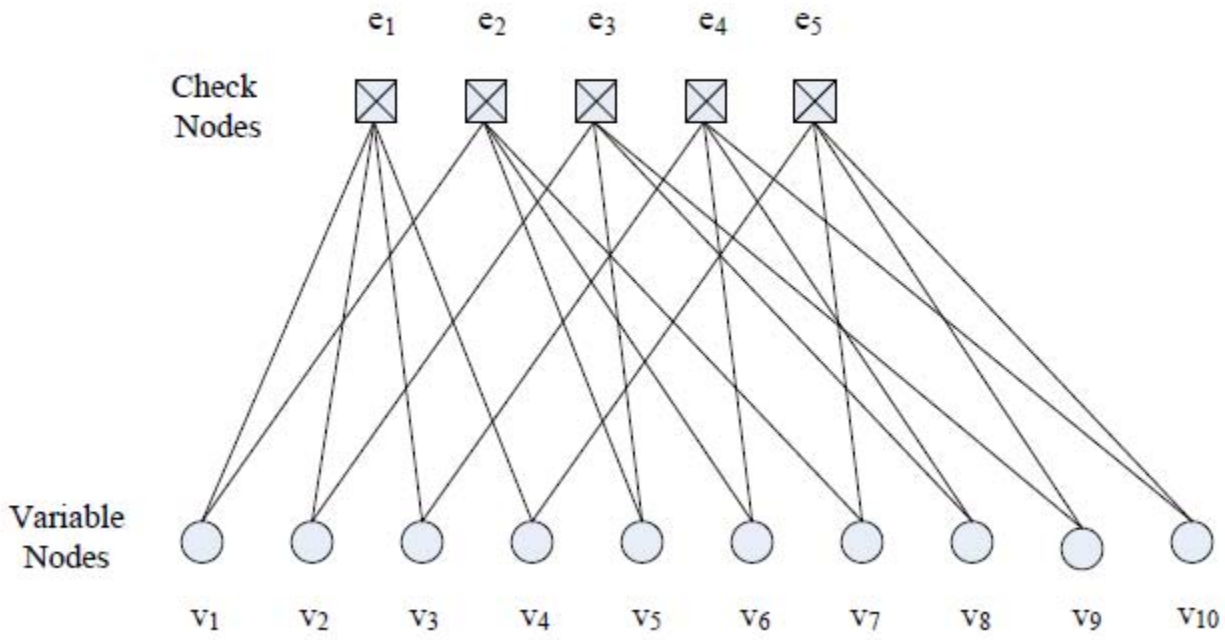


Figure 3.1. Tanner graph for example code.

tions. The concept of such algorithm is as follows:

The *a posteriori probability* (APP) that a given bit in the transmitted codeword $v = [v_1 \quad v_2 \quad \cdots \quad v_N]$ equals 1, given the received word $y = [y_1 \quad y_2 \quad \cdots \quad y_N]$ is computed initially. So, for the decoding of bit v_i, the APP is computed as follow

$$Pr(v_i = 1|y) \tag{3.5}$$

The APP ratio (also called likelihood ratio, LR) is given by

$$l(v_i) \triangleq \frac{Pr(v_i = 0|y)}{Pr(v_i = 1|y)}. \tag{3.6}$$

The more numerically stable computation of the log-APP ratio, also called the log-likelihood ratio (LLR) is given by:

$$L(v_i) \triangleq \log\left(\frac{Pr(v_i = 0|y)}{Pr(v_i = 1|y)}\right). \tag{3.7}$$

Hereafter, the natural logarithm is assumed for LLRs.

The MPA for the computation of $Pr(v_i = 1|y)$, $l(v_i)$, or $L(v_i)$ is an iterative algorithm which is based on the code's Tanner graph. Specifically, it is imagined that the variable nodes (VNs) represent processors of one type, check nodes (CNs) represent processors of another type, and the edges represent message paths. In one half iteration, each variable node processes its input messages and passes its resulting output messages up to neighboring check nodes (two nodes are said to be neighbors if they are connected by an edge). This is depicted in Figure 3.2 for the message $m_{\uparrow 14}$ from v-node v_1 to c-node e_4 (the subscripted arrow indicates the direction of the message, keeping in mind that our Tanner graph convention places c-nodes above v-nodes). The information passed concerns $Pr(v_1 = b \quad | \quad \text{input messages})$, $b \in \{0, 1\}$, the ratio of such probabilities, or the logarithm of the ratio of such probabilities. Note in the figure that the information passed to c-node e_4 is all the information available to v-node v_1 from the channel and through its neighbors, excluding c-node e_4; that is, only extrinsic information is passed. Such extrinsic information $m_{\uparrow ij}$ is computed for each connected v-node / c-node pair v_i/e_i at each half-iteration.

In the other half iteration, each c-node processes its input messages and passes its resulting output messages down to its neighboring v-nodes. This is depicted in Figure 3.3 for the message $m_{\downarrow 14}$ from c-node e_1 down to v-node v_4. The information passed concerns $Pr(\text{check equation } e_1 \text{ is satisfied} \quad | \quad \text{input messages})$, $b \in \{0, 1\}$, the ratio of such probabilities, or the logarithm of the ratio of such probabilities. Note, as for the previous case, only extrinsic information is passed to v-node v_4. Such extrinsic information $m_{\downarrow ji}$ is computed for each connected c-node/v-node pair e_j/v_i at each half-iteration.

After a prescribed maximum number of iterations or after some stopping criterion has been met, the decoder computes the APP, the LR, or the LLR from which decisions on the bits v_i are made.

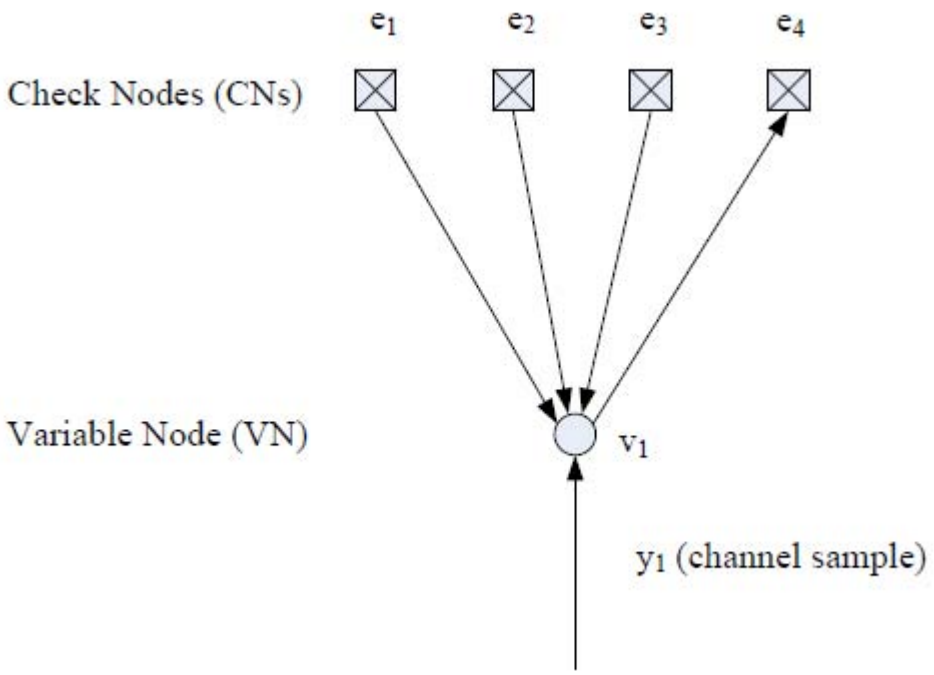


Figure 3.2. Subgraph of a Tanner graph corresponding to an H matrix whose first column is $[1 \quad 1 \quad 1 \quad 1 \quad 0 \quad \cdots \quad 0]$**. The arrows indicate message passing from node** v_1 **to node** e_4**.**

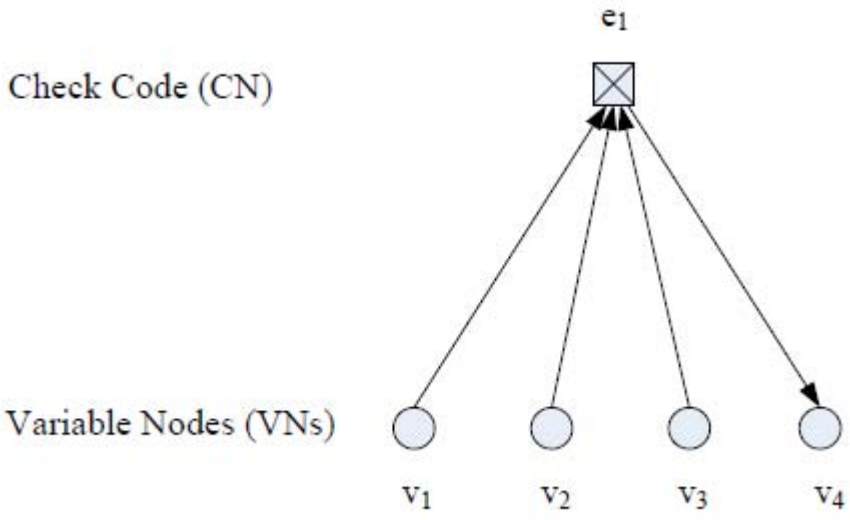


Figure 3.3. Subgraph of a Tanner graph corresponding to an H matrix whose first row is $[1 \quad 1 \quad 1 \quad 1 \quad 0 \quad \cdots \quad 0]$**. The arrows indicate message passing from node** e_1 **to node** v_4**.**

3.3 LT decoding using Iterative Decoding Algorithm

In noisy channel decoding, the message passing rules of fountain codes resemble those of LDPC decoder. We see that for LDPC codes, all the check nodes have value 0 but in LT codes the output values can be 0 or 1. Therefore, we need to create a H matrix (as one discussed in LDPC codes) such that every check node has value 0.

3.3.1 Creation of H matrix

To use the message passing algorithm for decoding the LT codes over AWGN channel, we developed a H matrix for LT codes. We have used the optimized degree distribution mentioned in Shokrollahi's paper [15] as our degree distribution to generate code symbols, which is as follows:

$$\begin{aligned}\Omega(x) = .008x + .494x^2 + .166x^3 + .073x^4 + .083x^5 + .056x^8 + .037x^9 \\ + .056x^{19} + .025x^{65} + .003x^{66}\end{aligned} \tag{3.8}$$

where the exponents of x denote the degrees of the input symbols and the corresponding coefficients denote the probability of choosing such a degree.

Let X represent the data symbols ($X = [x_1 \quad x_2 \quad \cdots \quad x_k]$) and C represent the code symbols ($C = [c_1 \quad c_2 \quad \cdots \quad c_n]$) for LT codes such that $C = XG$ where G is a generator matrix of size $k \times n$. These encoded symbols are transmitted over the AWGN channel. The tanner graph for this LT code is given in Figure 3.4.

The values of data symbols are either 0 or 1 and the values of code symbols are also either 0 or 1 as the encoded symbols are created by the XORing of the randomly selected data symbols. Now, we need to create variable nodes and check nodes such that the sum of the values of the variable nodes being connected to a corresponding check node in a Tanner graph should equal to zero. To do this, we first aligned both the data symbols X and code symbols C horizontally to get VNs as shown in Figure 3.5 and then created the corresponding CNs. Let V be the horizontal concatenation of X and C such that

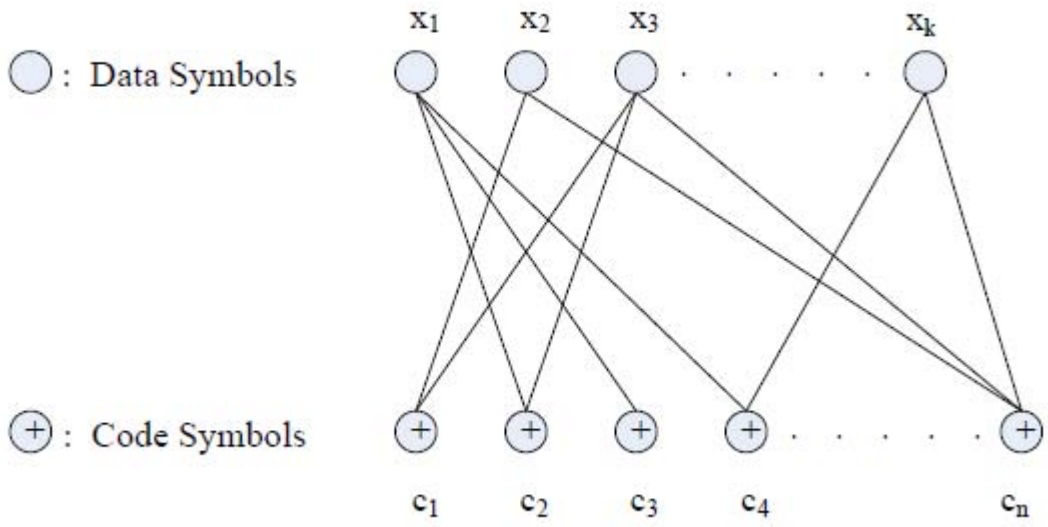


Figure 3.4. Tanner graph of LT codes

$V = [X \quad C]$, that is, $V = [x_1, x_2, \cdots x_k, c_1, c_2, \cdots c_n]$. We see that the tanner graph of LT codes contain two kinds of variable nodes as shown in Figure 3.5. One kind is the information variable nodes $X = [x_1 \quad x_2 \quad \cdots \quad x_k]$ which are not transmitted and have no channel information. Another is encoding variable nodes $C = [c_1 \quad c_2 \quad \cdots \quad c_n]$ which are transmitted over the channel and contain channel information.

Let N be the total number of VNs and $N = k + n$. Once we create our VNs and CNs, the H matrix can be developed which will be the same as the one we get by the horizontal concatenation of the transpose of G matrix and the identity matrix of size $n \times n$ as following:

$$H = \left[G^T, I\right] \tag{3.9}$$

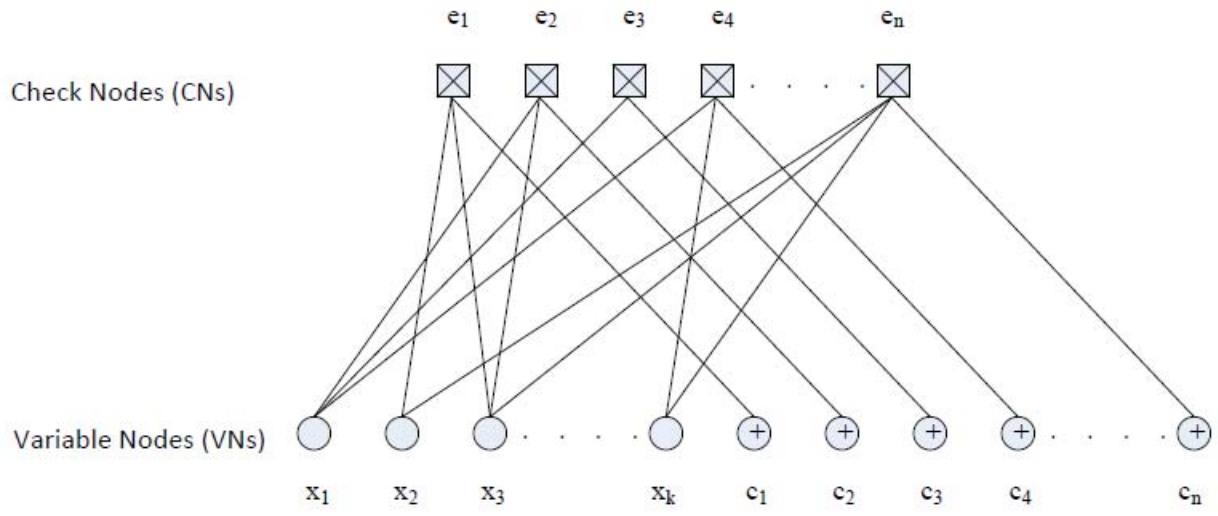


Figure 3.5. Tanner graph showing VNs and CNs for LT codes over AWGN channel

3.3.2 Calculation of $L(v_i)$ for BI-AWGN channel

The initial input to the decoder is the log-likelihood ratio (LLR), $L(v_i)$, which is defined by the following equation:

$$L(v_i) = log\left(\frac{Pr(v_i = 0 \quad | \quad \text{channel output for} \quad v_i)}{Pr(v_i = 1 \quad | \quad \text{channel output for} \quad v_i)}\right) \tag{3.10}$$

The first k symbols of the V are the source symbols and they are not transmitted over the channel. Only the encoded symbols are transmitted over the channel and are received at the receiver. So, for $i = 1, 2, \cdots, k, \quad L(v_i) = 0$ and for all the received symbols, $L(v_i)$ is calculated by using (3.10).

When transmitting a binary codeword on the BI-AWGN channel, the codeword bits $v_i \in \{0, 1\}$ can be mapped to the symbols $x_i \in \{-1, +1\}$ in one of two ways: $\{0 \to 1, 1 \to -1\}$ or $\{0 \to -1, 1 \to 1\}$. We use the traditional convention $\{0 \to 1, 1 \to -1\}$.

The received LLRs for the BI-AWGN channel are then

$$\begin{aligned} L(v_i) = L(x_i|y_i) &= log\frac{p(v_i=0|y_i)}{p(v_i=1|y_i)} \\ &= log\frac{p(x_i=1|y_i)}{p(x_i=-1|y_i)} \\ &= log\frac{p(y_i|x_i=1)p(x_i=1)/p(y_i)}{p(y_i|x_i=-1)p(x_i=-1)/p(y_i)} \\ &= log\frac{p(y_i|x_i=1)p(x_i=1)}{p(y_i|x_i=-1)p(x_i=-1)}. \end{aligned} \tag{3.11}$$

where we have used Bayes' rule as defined below:

$$p(x_i|y_i) = p(x_i, y_i)/p(y_i) = p(y_i|x_i)p(x_i)/p(y_i). \tag{3.12}$$

to substitute for $p(x_i = 1/y_i)$ and $p(x_i = -1/y_i)$. If the source is equiprobable then $p(x_i = -1) = p(x_i = 1)$, and we have,

$$L(v_i) = L(x_i|y_i) = log\frac{p(y_i|x_i=1)}{p(y_i|x_i=-1)}. \tag{3.13}$$

For the BI-AWGN channel:

$$p(y_i|x_i=1) = \frac{1}{\sqrt{2\pi\sigma^2}}\exp\left(-\frac{(y_i-\mu)^2}{2\sigma^2}\right), \tag{3.14}$$

$$p(y_i|x_i=-1) = \frac{1}{\sqrt{2\pi\sigma^2}}\exp\left(-\frac{(y_i+\mu)^2}{2\sigma^2}\right); \tag{3.15}$$

thus

$$\begin{aligned} L(v_i) = L(x_i|y_i) &= log\frac{\frac{1}{\sqrt{2\pi\sigma^2}}\exp\left(-\frac{(y_i-\mu)^2}{2\sigma^2}\right)}{\frac{1}{\sqrt{2\pi\sigma^2}}\exp\left(-\frac{(y_i+\mu)^2}{2\sigma^2}\right)} \\ &= log\exp\left(-\frac{(y_i-\mu)^2}{2\sigma^2}+\frac{(y_i+\mu)^2}{2\sigma^2}\right) \\ &= \frac{1}{2\sigma^2}(-(y_i^2-2\mu y_i+\mu^2)+(y_i^2+2\mu y_i+\mu^2)) \\ &= \frac{2\mu}{\sigma^2}y_i. \end{aligned} \tag{3.16}$$

The LLR value for a bit v_i is sometimes called a *soft decision* for v_i. A *hard decision* for v_i will return $v_i = 0$, equivalently $x_i = 1$, if $L(v_i)$ is positive and $v_i = 1$, equivalently $x_i = -1$, if $L(v_i)$ is negative.

When considering the relative noise level of a BI-AWGN channel, it is convenient to assume that $\mu = 1$ and adjust σ to reflect the noise quality of the channel. In this case $L(v_i)$ can be written as:

$$L(v_i) = \frac{2}{\sigma^2}y_i. \tag{3.17}$$

3.3.3 Decoding Algorithm

Once the H matrix is created we can apply the iterative decoding algorithms used in LDPC codes [20].

The other three key variables other than $L(v_i)$ used in the decoding algorithm are: $L(r_{ji})$, $L(q_{ij})$, and $L(Q_i)$ where, $L(r_{ji})$ represents the outgoing CN message from check node j to variable node i and is calculated for all the check nodes, $L(q_{ij})$ represents the outgoing VN message from variable node i to check node j and is calculated for all the variable nodes and $L(Q_i)$ is the final LLR value computed for every variable node from which final decision on source symbols will be made. These three key variables are given by the following equations:

$$L(r_{ji}) = 2\tanh^{-1}\left(\prod_{i'\in N(j)-\{i\}}\tanh\left(\frac{1}{2}L(q_{i'j})\right)\right) \tag{3.18}$$

$$L(q_{ij}) = L(v_i) + \sum_{j' \in N(i) - \{j\}} L(r_{j'i}) \tag{3.19}$$

$$L(Q_i) = L(v_i) + \sum_{j' \in C_i} L(r_{j'i}) \tag{3.20}$$

Once we have these equations, we can apply the following algorithm for decoding:

1. **Initialization:** For all i, initialize $L(v_i)$ for the appropriate channel model. Then, for all i, j for which $h_{ji} = 1$, set $L(q_{ij}) = L(v_i)$.

2. **CN update:** Compute outgoing CN messages $L(r_{ji})$ for each CN and then transmit to the VNs.

3. **VN update:** Compute outgoing VN messages $L(q_{ij})$ for each VN and then transmit to the CNs.

4. **LLR total:** For $i = 1, 2, \cdots, N$, compute $L(Q_i)$.

5. **Stopping criteria:** For $i = 1, 2, \cdots, N$, set

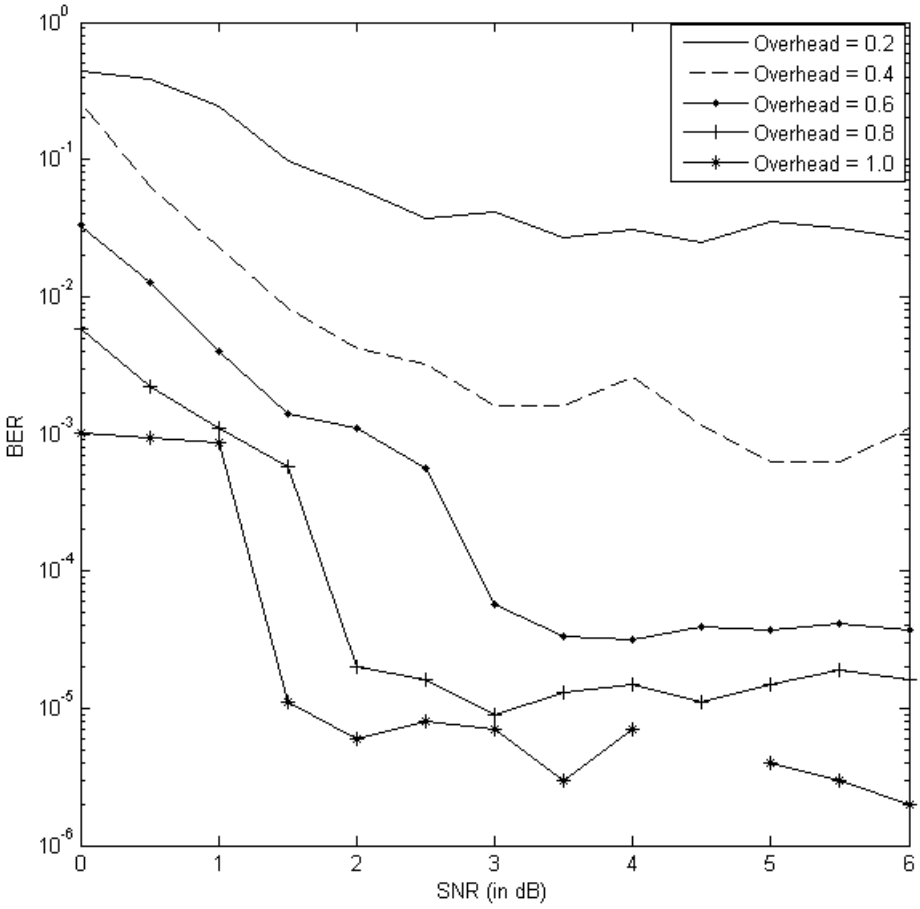
$$\hat{v} = \begin{cases} 1, & \text{if} \quad L(Q_i) < 0 \\ 0, & \text{otherwise} \end{cases}$$

to obtain $\hat{v}$. If $\hat{v}H^T = 0$ or the number of iterations equals the maximum limit, stop; else, go to Step 2.

3.4 Simulation Results and Discussion

All the simulations were done using C/C++ programming. The original data is encoded using the LT encoding algorithm. The simulation is performed 1000 times for a data block of 1000 bits. The encoded symbols are transmitted over the AWGN channel and collected at the receiver where decoding algorithm is applied over a number of iterations on the Tanner

graph generated from the H-matrix to calculate the LLRs of input symbols. Based on these LLRs, whether the input symbol is "0" or "1" is determined. Simulation results are shown in Figures 3.6 and 3.7 where the dependence of bit error rate (BER) on signal to noise ratio (SNR) is shown. The value of SNR (E_b/N_0) changes from 0 to 6 dB. The number of iterations used is 50. Overhead is the fraction of extra symbols received at the receiver and this value is varied from 0.2 to 1.0 at an interval of 0.2.



he

Figure 3.6. LT Decoding over AWGN channel using the degree distribution optimized for LT codes used in Raptor codes.

Observing both figures, it is clear that as the SNR increases, the BER decreases. Also, as the overhead increases, the BER again goes on decreasing. Even at the higher SNR the

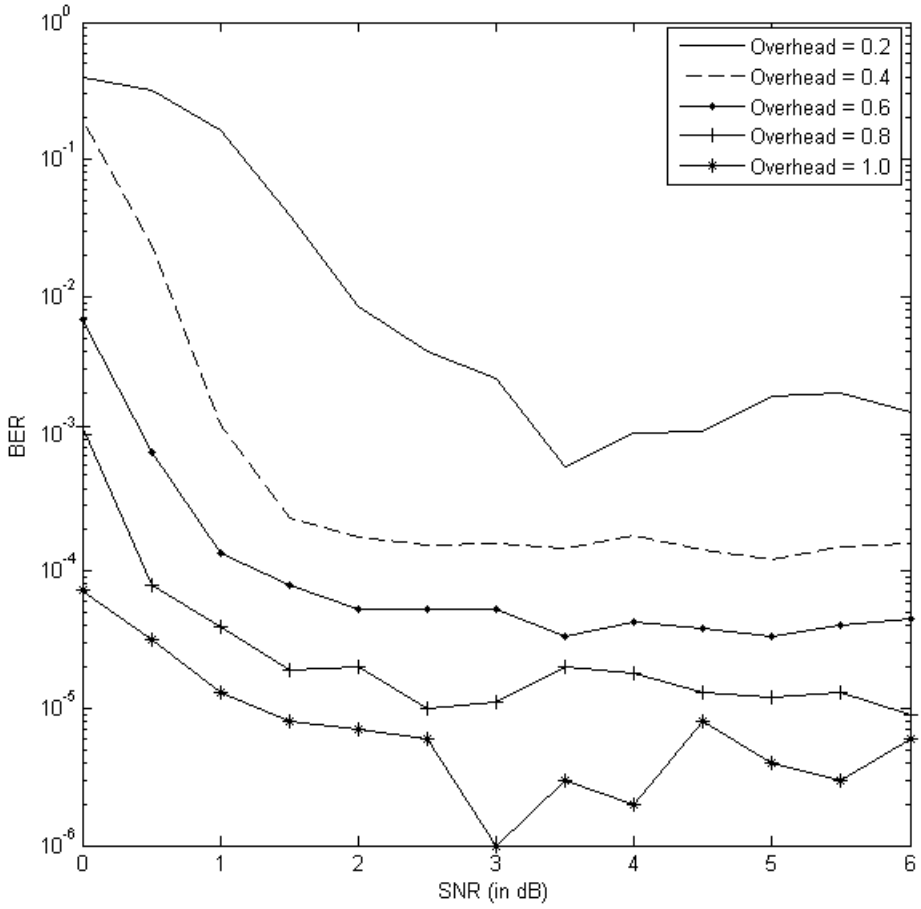


Figure 3.7. LT Decoding over AWGN channel using revised degree distribution

BER is not impressive and also with increasing overhead the improvement in BER in not impressive either. This suggests that the LT codes alone are not good at correcting errors.

Regarding Figure 3.6, the distribution used while encoding the input symbols at the source is the one given in (3.8). This distribution was used as an optimized degree distribution for LT codes while using in Raptor codes which was suitable for a higher data block size. In our case, as the data block size is 1000 bits only and the probability for generating degree one symbols is low (just 0.8%), there are possibilities that degree one symbols are never generated and decoding never starts, which leads to a poor performance. Therefore, to make sure that degree one symbols are always generated and decoding never fails to start,

we revised the distribution and increased the probability of generating degree one symbols to 2% and the probabilities of generating other degree symbols are adjusted accordingly. The revised distribution is as follows:

$$\begin{aligned}\Omega(x) = .020x + .488x^2 + .159x^3 + .073x^4 + .083x^5 + .056x^8 + .037x^9 \\ + .056x^{19} + .025x^{65} + .003x^{66}\end{aligned} \tag{3.21}$$

This revised degree distribution was used for encoding data symbols while generating graph depicted in Figure 3.7 where we can clearly see the improvements in the curve as compared to those in Figure 3.6. Thus, it can be inferred that the performance of LT codes over AWGN channel can be improved by optimizing the degree distribution and this requires further research.

Chapter 4

Turbo Codes

4.1 Introduction

In 1949 Claude Shannon published a classic paper [21] that established a mathematical basis for the consideration of the noisy communications channel. In his analysis he quantified the maximum theoretical capacity for a communication channel, the Shannon limit, and indicated that error-correcting channel codes exist that allowed this maximum capacity to be achieved.

In 1993 Berrou, Glavieux and Thitimajshima [22] proposed a new class of convolution codes called turbo codes whose performance in terms of bit error rate (BER) are close to the Shannon limit. They indicated that it was possible to operate within 0.7 dB of the Shannon limit. Finally, advancements and high demands in the fields of mobile, satellite and space communication systems soon realized turbo codes as a channel coding standard in many modern day areas of communication.

4.2 Principles of Turbo Codes

It is theoretically possible to approach the Shannon limit by using a block code with large block length or a convolutional code with a large constraint length. The processing power required to decode such long codes makes this approach impractical. Turbo codes overcome this limitation by using recursive coders and iterative decoders. Most frequently, a turbo code refers to a concatenation of two (or more) convolutional encoders separated

by interleavers. The turbo decoder consists of two (or more) soft-in/soft-out convolutional decoders which iteratively feed probabilistic information back and forth to each other. The recursive coder makes convolutional codes with short constraint length appear to be block codes with a large block length, and the iterative soft decoder progressively improves the estimate of the received message.

4.3 Convolutional Codes

Convolutional codes [20], invented by Elias in 1955 [23] are linear codes with a very distinct algebraic structure. Multiple convolutional codes are concatenated to get turbo (-like) codes. The encoders of convolutional codes are usually described as stream-oriented although they can be utilized in block-oriented situations. In contrast with a block code, whose encoder assigns an n-bit codeword to each block of k data bits, a convolutional encoder assigns code bits to an incoming information bit stream continuously, in a stream-oriented fashion.

An example of classical non systematic convolutional encoder with rate -1/2 is depicted in Figure 4.1. Two code bits are produced for each data bit that enters the encoder; hence, the rate is 1/2. The contents of the four binary memory cells (D) define the state of the encoder; hence, the encoder is a 16-state device. The two adders are the modulo-2 adders. When a feedback is presented in the encoder realization and when one of the encoder output is the copy of input itself, then the code is termed as systematic recursive code. Thus, Figure 4.2 is an example of recursive systematic convolutional (RSC) encoder where the output y_k is equal to the input bit u_k.

4.4 Turbo Encoder

The classical turbo-code encoder is a rate $R = 1/3$ parallel concatenated code composed of two binary rate-1/2 recursive systematic convolutional codes separated by a K-bit interleaver or permuter, together with an puncturing mechanism. The overall turbo encoder

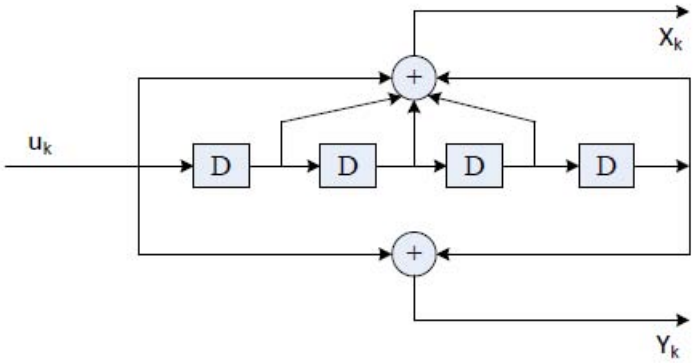


Figure 4.1. Classical Non-Systematic Convolutional Encoder.

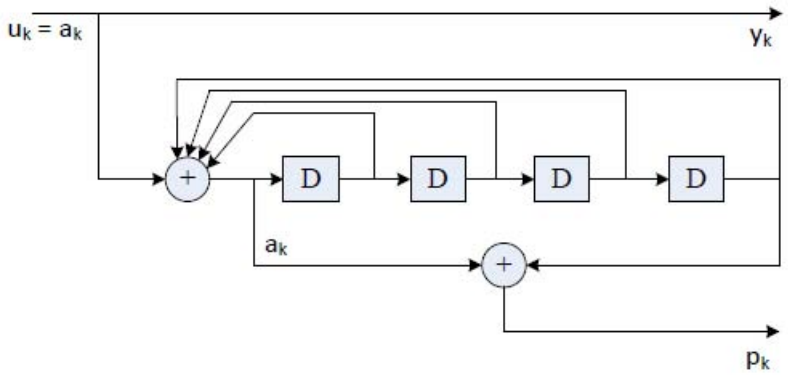


Figure 4.2. Recursive Systematic Convolutional Encoder

produces a codeword for an input sequence u_k consisting of K bits by combining this message sequence, which is often called the systematic bit y_k together with the two parity sequences p_{1k} and p_{2k}, which are obtained by encoding the message stream with two identical encoders to produce an output encoded stream. The second encoder operates on the interleaved version of the message sequence in order to produce a different output stream. Therefore, the interleaver is an important design parameter in the performance evaluation of turbo codes. Clearly, without the puncturer, the encoder is rate 1/3, mapping K data bits to $3K$ code bits. The general structure for the classical rate 1/3 turbo encoder is shown in Figure 4.3. Figure 4.4 is the detailed version of Figure 4.3, where two identical RSC codes with parallel concatenation is shown. Both elementary encoder (C_1 and C_2) inputs use the same bit u_k

but according to a different sequence due to the presence of an interleaver. For an input bit sequence $\{u_k\}$, encoder outputs x_k and y_k at time k are respectively equal to u_k and interleaved version of u_k (systematic encoder) and to encoder C_1 output p_{1k}, or to encoder C_2 output p_{2k} which are the parity bit sequence. The systematic output of Encoder-2 is not transmitted because this is just the interleaved version of systematic output of Encoder-1.

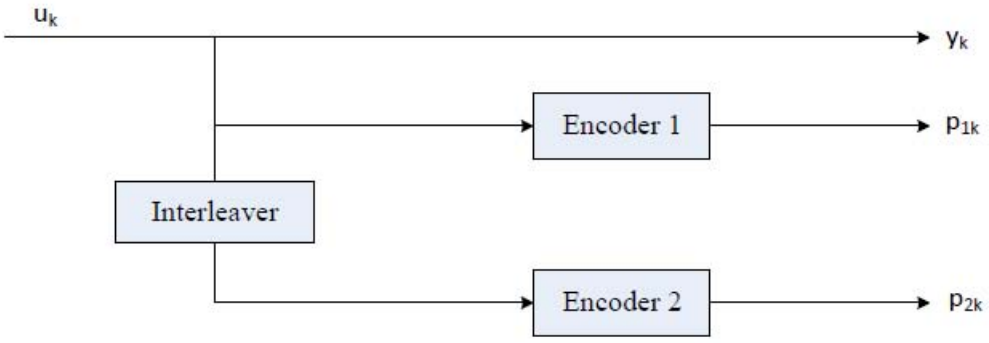


Figure 4.3. A general rate 1/3 turbo encoder

4.5 Puncturing

Puncturing [20] is the process of achieving higher code rates from a lower-rate code, which means that code bits at the encoder output are periodically deleted and hence not transmitted. For example, to achieve a rate 2/3 convolutional code by puncturing a rate-1/2 encoder, one need only puncture every fourth code bit at the encoder output. Since each group of four code bits is produced by two information bits, but only three of the four will be sent, rate 2/3 has been achieved, that is, overhead is reduced.

Similarly, to get a rate-1/2 for the a general 1/3 turbo encoder shown in Figure 4.3, odd bits coming from the Encoder 1 can be punctured and even bits coming from Encoder 2 can be punctured.

Generally speaking, puncturing can set the code rate to an arbitrary value, not just 1/2 for the above example. Let N be the size of the message block. Let n_1 bits per block from Encoder 1 and n_2 bits per block from Encoder 2 are chosen, that is, rest of others are

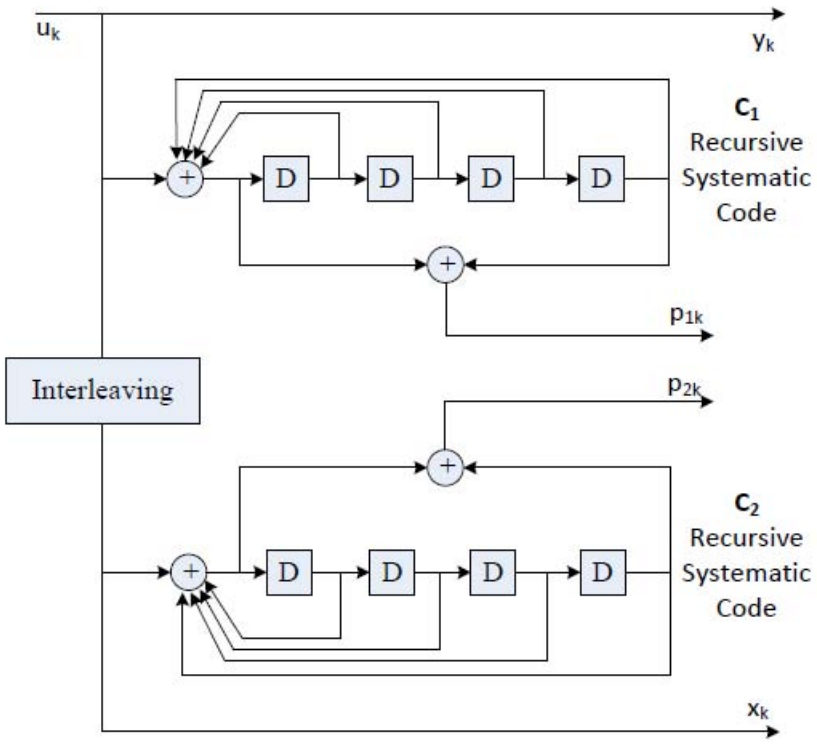


Figure 4.4. Recursive Systematic Codes with parallel concatenation.

punctured, then the two elementary code rates R_1 and R_2 associated with Encoder 1 and Encoder 2 are:

$$R_1 = \frac{N}{n_1 + N} \tag{4.1}$$

and

$$R_2 = \frac{N}{n_2 + N} \tag{4.2}$$

The overall rate R can be expressed as suggested in [24] as:

$$\begin{aligned} \frac{1}{R} &= \frac{1}{R_1} + \frac{1}{R_2} - 1 \\ &= \frac{R_1 + R_2 - R_1 R_2}{R_1 R_2}. \end{aligned} \tag{4.3}$$

where R_1 and R_2 are different, and if $R_1 \leq R_2$, optimum decoding performance can be

obtained [24].

4.6 Interleaving

Interleaving [20] is the process of taking each incoming block of bits and rearranging them prior to encoding by the second RSC encoder. Conventionally, interleaving is used to spread bursts of errors over multiple code-words that can be corrected by error-correcting codes. By converting bursts of errors into random-like errors, it thus becomes an effective means to combat error bursts. For turbo codes, the interleaver has more functions. Interleaving is used to feed the encoders with permutations so that the generated redundancy sequences can be assumed independent. Some interleaver types used in turbo codes are: "Row-Column" interleavers, "Helical" interleavers, "Odd-even" interleavers,"Simile" interleavers, "Frame" interleavers, "Pseudo-random" interleavers, "S-type" interleavers, "Uniform" interleavers, which are discussed in [25].

4.7 Turbo Decoder

The structure of the turbo decoder is shown in Figure 4.5. It consists of a pair of decoders which work cooperatively to refine and improve the estimate of the original information bits. The decoders are based on the MAP (maximum a-posteriori probability) algorithm and output soft decision information learned from the corrupted parity bits. Initially decoder-1 starts without initial information (a-priori estimates are set to zero). In subsequent iterations, the soft decision information of one decoder is used to initialize the other decoder. The decoder information is cycled around the loop until the soft decisions converge on a stable set of values. The latter soft decisions are then sliced to recover the original binary sequence.

Lets start by summarizing what is meant by the terms *a-priori*, *a-posteriori*, and *extrinsic information* which are central concepts behind the iterative decoding of turbo codes.

- **a-priori** The a-priori information about a bit is the information known before decoding starts, from a source other than the received sequence or the code constraints. It is also sometimes referred to as intrinsic information to contrast with the extrinsic information described next.

- **extrinsic** The extrinsic information about a bit u_k is the information provided by a decoder based on the received sequence and on a-priori information excluding the received systematic bit and the a-priori information $L(u_k)$ for the bit u_k. Typically, the component decoder provides this information using the constraints imposed on the transmitted sequence by the code used. It processes the received bits and a-priori information surrounding the systematic bit u_k, and uses this information and the code constraints to provide information about the value of u_k.

- **a-posteriori** The a-posteriori information about a bit is the information that the decoder gives taking into account all available sources of information about u_k. It is the a-posteriori LLR, that is, $L(u_k|y')$, that MAP algorithm gives as its output.

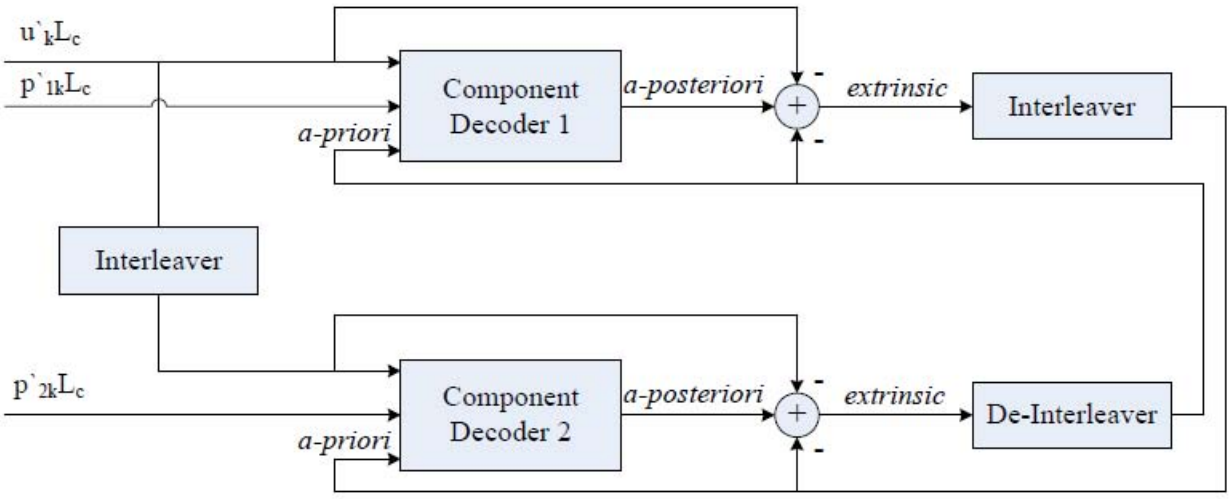


Figure 4.5. Turbo decoding schematic.

As seen in the figure, each decoder takes three inputs: 1) the systematically encoded channel output bits; 2)the parity bits transmitted from the associated component encoder; and 3) the information from the other component decoder about the likely values of the bits concerned. This information from the other decoder is referred to as a-priori information. The component decoders have to exploit both the inputs from the channel and this a-priori information. They must also provide what are known as soft outputs for the decoded bits. This means that as well as providing the decoded output bit sequence, the component decoders must also give the associated probabilities for each bit that it has been correctly decoded.

The soft outputs from the component decoders are typically represented in terms of the so-called log likelihood ratios (LLRs), the magnitude of which gives the sign of the bit, and the amplitude the probability of a correct decision. The LLR $L(u_k)$ (a-priori information)for the value of a decoded bit u_k is given by:

$$L(u_k) = \log\left(\frac{P(u_k = +1)}{P(u_k = -1)}\right) \tag{4.4}$$

where $P(u_k = +1)$ is the probability that the bit $u_k = +1$, and similarly for $P(u_k = -1)$.

4.7.1 MAP Decoding

The MAP algorithm minimizes the probability of bit error by using the entire received sequence to identify the most probable bit at each state of the trellis. It provides not only the estimated bit sequence, but also the probabilities for each bit that has been decoded correctly. This algorithm has also become known as BCJR algorithm [26], named after its inventors.

Lets use the following shorthand for the transmitted and received symbol pairs:

$$y_k = \{u_k, p_k\} \quad y = y_{1,K} = \{y_1, y_2, \cdots, y_k\} \qquad y_k^{'} = \{u_k^{'}, p_K^{'}\} \quad y' = y'_{1,K} = \{y'_1, y'_2, \cdots, y'_K\}$$

The MAP algorithm gives, for each decoded bit u_k, the probability that this bit was +1 or -1, given the received symbol sequence y'. This is equivalent to finding the a-posteriori LLR $L(u_k/y')$ given below:

$$L(u_k/y') = \log\left(\frac{P(u_k = +1|y')}{P(u_k = -1|y')}\right). \tag{4.5}$$

As mentioned in [27], for a convolutional code, the likelihood ratio can be expressed in terms of the trellis. If the previous state $S_{k-1} = s'$ and the present state $S_k = s$ are known in a trellis then the input bit u_k which caused the transition between these states will be known. This, along with Bayes rule and the fact that the transitions between the previous state and the present state in a trellis are mutually exclusive (that is, only one of them could have occurred at the encoder) allow us to write the following:

$$\frac{P(u_k = +1|y')}{P(u_k = -1|y')} = \frac{\sum_{(s',s)\in S^+} P(s_{k-1} = s', s_k = s|y')}{\sum_{(s',s)\in S^-} P(s_{k-1} = s', s_k = s|y')} = \frac{\sum_{(s',s)\in S^+} P(s_{k-1} = s', s_k = s, y')}{\sum_{(s',s)\in S^-} P(s_{k-1} = s', s_k = s, y')} \tag{4.6}$$

The numerator sum is over all possible state transitions associated with a "+1" data bit, and the denominator over all possible state transitions associated with a "-1" data bit.

The probability of a particular state transition and the noisy observation associated with a trellis transition can be expressed (using Bayes theorem) as:

$$\begin{aligned} P(s_{k-1} = s', s_k = s, y'_{1,K}) &= P(s_{k-1} = s', y'_{1,k-1}) \cdot P(s_k = s, y'_k|s_{k-1} = s') \cdot P(y'_{k+1,K}|s_k = s) \\ &= \alpha_{k-1}(s') \cdot \gamma_k(s', s) \cdot \beta_k(s) \end{aligned} \tag{4.7}$$

The probabilities associated with the continuous valued received observation taking a particular value are infinitesimally small. As the final result will be a probability ratio (the likelihood ratio) we can relax the notation and work with probabilities to help keep the

mathematics simple.

In terms of the above definitions of $\alpha_{k-1}(s'), \gamma_k(s', s), \beta_k(s)$, the a-posteriori likelihood ratio can be rewritten as the ratio as in [27] as follows:

$$\frac{P(u_k = +1|y')}{P(u_k = -1|y')} = \frac{\sum_{(s',s)\in S^+} \alpha_{k-1}(s') \cdot \gamma_k(s', s) \cdot \beta_k(s)}{\sum_{(s',s)\in S^-} \alpha_{k-1}(s') \cdot \gamma_k(s', s) \cdot \beta_k(s)} \tag{4.8}$$

The term $\alpha_k(s')$ is the probability of arriving at a branch in a particular state and the sequence of noisy observations $y'_{1,k} = y'_1, y'_2, \cdots, y'_k$ which led up to that state. By summing over all paths leading into that state we get a *forward recursion* for calculating $\alpha_k(s')$ in terms of $\gamma_k(s', s)$.

$$\begin{aligned} \alpha_k(s) = P(s_k = s, y'_{1,k}) &= \sum_{s'} P(s_{k-1} = s', y'_{1,k-1}) \cdot P(s_k = s, y'_k | s_{k-1} = s') \\ &= \sum_{s'} \alpha_{k-1}(s') \cdot \gamma_k(s', s) \end{aligned} \tag{4.9}$$

To begin the forward recursion, the forward state probabilities should be initialized. In turbo coders all convolutional encoders are started in state 0. Thus the forward recursion is started with:

$$\alpha_0(s) = P(s_0 = s) \begin{cases} 1 & \text{for} \quad s = 0 \\ 0 & \text{for} \quad s \neq 0 \end{cases} \tag{4.10}$$

The term $\beta_k(s)$ is the probability of exiting a branch via a particular state s and the sequence of noisy observations $y'_{1,k} = y'_1, y'_2, \cdots, y'_k$ which finish off the trellis. By summing over all paths exiting that state we get a *backward recursion* for calculating $\beta_k(s)$ in terms of values

$\gamma_k(s', s)$ as following:

$$\begin{aligned}\beta_k(s') = P(y'_{k+1,K}|s_k = s) &= \sum_s P(s_{k+1} = s, y'_{k+1}|s_k = s') \cdot P(y'_{k+2,K}|s_k = s') \\ &= \sum_s \gamma_{k+1}(s', s) \cdot \beta_{k+1}(s)\end{aligned} \tag{4.11}$$

To begin the backward recursion, the backward state probabilities should be initialized. Convolutional encoder#1 is usually terminated at state 0. However, in general, the final state for convolutional encoder#2 is data dependent, and unknown beforehand. So, an uniform distribution for the final state of encoder#2 is assumed. A suitable initialization (where the number of states in the convolutional encoder is 2^v, v being the memory cells used) is as follows:

$$\text{MAP \#1}: \qquad \beta_K(s) = P(s_K = s) \begin{cases} 1 & \text{for} \quad s = 0 \\ 0 & \text{for} \quad s \neq 0 \end{cases}$$

$$\text{MAP \#2}: \qquad \beta_K(s) = P(s_K = s) = \frac{1}{2^v} \quad \text{for all} \quad s$$

For a given state transition, the transmitted signal is the data bit and parity pair y_k. Also, for a given starting state, the next state is completely determined by the value of the data bit. Using the Bayes theorem, the branch probability can be expressed as following:

$$\gamma_k(s', s) = P(s_k = s, y'_k|s_{k-1} = s') = P(y'_k|s_{k-1} = s', s_k = s) \cdot P(s_k = s|s_{k-1} = s') = P(y'_k|y_k) \cdot P(u_k) \tag{4.12}$$

The probability of the data bit taking a particular value can be expressed in terms of the

log likelihood of the a-priori probability ratio as

$$\begin{aligned} P(u_k) &= \frac{\exp\left[\frac{1}{2}L_a(u_k)\right]}{1+\exp[L_a(u_k)]} \cdot \exp\left[\frac{1}{2}u_k L_a(u_k)\right] \\ &= B_k \cdot \exp\left[\frac{1}{2}u_k L_a(u_k)\right] \end{aligned} \tag{4.13}$$

where $L_a(u_k) = \log\left[\frac{P(u_k=+1)}{P(u_k=-1)}\right]$

The probability of the observed noisy data bit and parity symbols taking particular values can be expressed in terms of Gaussian probability distribution as

$$\begin{aligned} P(y'_k|y_k) &= P(u'_k|u_k) \cdot P(p'_k|p_k) \\ &= \frac{1}{\sqrt{2\pi\sigma^2}} \cdot \exp\left[\frac{-(x'_k - x_k)^2}{2\sigma^2}\right] \cdot \Delta \cdot \frac{1}{\sqrt{2\pi\sigma^2}} \cdot \exp\left[\frac{-(p'_k - p_k)^2}{2\sigma^2}\right] \cdot \Delta \\ &= A_k \cdot \exp\left[\frac{u'_k u_k - p'_k p_k}{\sigma^2}\right] \end{aligned} \tag{4.14}$$

Here, the Δ terms are infinitesimally small ranges about the particular values. These terms will cancel out in the final expressions for the likelihood ratios.

Now, the transition probability $\gamma_k(s', s)$ can be expressed in terms of log likelihood ratios and the noisy observations as

$$\gamma_k(s', s) = A_k \cdot B_k \cdot \exp\left[\frac{1}{2}(u_k L_a(u_k) + u_k L_c u'_k + p_k L_c p'_k)\right] \tag{4.15}$$

where $L_c = \frac{2}{\sigma^2}$ is the channel information.

Recalling the definition of the MAP log likelihood ratio:

$$L(u_k) = \log\left[\frac{P(u_k=+1|y')}{P(u_k=-1|y')}\right] = \log\left[\frac{P(y'_k|u_k=+1)}{P(y'_k|u_k=-1)}\right] + \log\left[\frac{P(u_k=+1)}{P(u_k=-1)}\right] \tag{4.16}$$

Now, we can write the following:

$$\begin{aligned} L(u_k) &= \log\left[\frac{\sum_{(s',s)\in S^+}\alpha_{k-1}(s')\cdot\gamma_k(s',s)\cdot\beta_k(s)}{\sum_{(s',s)\in S^-}\alpha_{k-1}(s')\cdot\gamma_k(s',s)\cdot\beta_k(s)}\right] \\ &= \log\left[\frac{\sum_{s',s\in S^+}\exp\left(\frac{L_a(u_k)}{2}\right)\cdot\exp\left(\frac{L_c u'_k}{2}\right)\cdot\alpha_{k-1}(s')\cdot\exp\left(\frac{p_k L_c p'_k}{2}\right)\cdot\beta_k(s)}{\sum_{s',s\in S^-}\exp\left(\frac{-L_a(u_k)}{2}\right)\cdot\exp\left(\frac{-L_c u'_k}{2}\right)\cdot\alpha_{k-1}(s')\cdot\exp\left(\frac{p_k L_c p'_k}{2}\right)\cdot\beta_k(s)}\right] \end{aligned} \tag{4.17}$$

Noting that the summation in the numerator is over all state transitions associated with a data bit symbol u_k equal to +1, and that the summation in the denominator is over all state transitions associated with a data bit symbol u_k equal to -1, we have:

$$L(u_k) = L_a(u_k) + L_c u'_k + \log\left[\frac{\sum_{(s',s)\in S^+}\alpha_{k-1}(s')\cdot\exp\left(\frac{p_k L_c p'_k}{2}\right)\cdot\beta_k(s)}{\sum_{(s',s)\in S^-}\alpha_{k-1}(s')\cdot\exp\left(\frac{p_k L_c p'_k}{2}\right)\cdot\beta_k(s)}\right] \tag{4.18}$$

Now, the MAP log likelihood can be separated into three distinct components as following:

$$L(u_k) = L_a(u_k) + L_c u'_k + L_e(u_k) \tag{4.19}$$

The first term $L_a(u_k)$ is the *a-priori* information. This will the initial estimate prior to running the MAP algorithm.

The second term $L_c u'_k$ is the information provided by that part of the noisy observation which does not depend on the convolutional code constraints.

The third term $L_e(u_k)$ is the information learned via the parity constraint. This information is referred to as *extrinsic* information.

In a turbo decoder the extrinsic information of one MAP decoder is used as the a-priori input to the other MAP decoder and is ping ponged back and forth between MAP decoders in each iterations. The iterative process continues, and with each iteration on average the BER of the decoded bits will improve.

Chapter 5

Turbo + LT codes

Turbo codes are high-performance forward error correction codes and are the first practical codes which have shown performance close to the Shannon limit. They are very efficient over noisy channel and are physical layer codes. On the other hand, LT codes are the first class of practical fountain codes that are near-optimal erasure correcting codes. It has been found that over noisy channels, the performance of LT codes is not impressive. Therefore, the objective here is to integrate LT codes and turbo codes and study the overall performance of the integrated system. Hence, the information at the source is first encoded by using turbo encoder followed by LT encoder and are transmitted over the noisy channel. Over noisy channels, errors may be introduced into individual symbols and at the receiver corrupted symbols are obtained which are processed by LT decoder followed by turbo decoder and a final decision about the obtained soft information is made. We would like to see whether the errors left after LT decoding can be corrected by using turbo decoding. The details of the method is explained in the following section.

5.1 System Model

The system model for our work is shown in Figure 5.1. Two layers of encoding of source information are done at the transmitting side. The source information which are the streams with binary bits, are at first encoded by using turbo encoder. We have used the rate compatible punctured turbo (RCPT) codes with a rate of 4/5. The turbo encoded symbols are then encoded by using LT encoder. The distribution used for LT encoding is not RSD

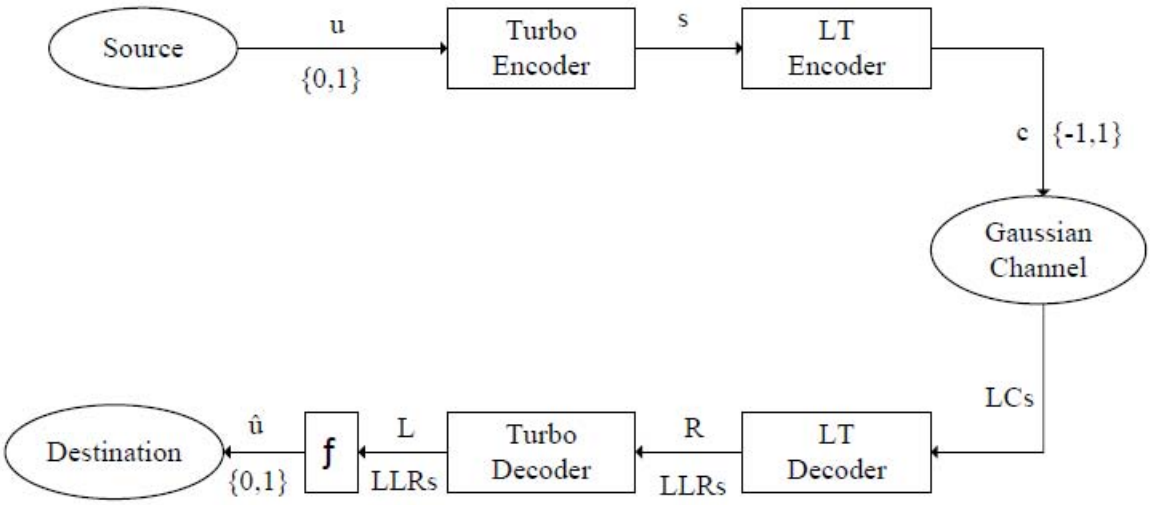


Figure 5.1. A system model for turbo + LT encoding decoding over BI-AWGN channel.

but the one mentioned in (3.8), which is optimized for LT codes while using in raptor codes where LDPC codes are considered but not turbo codes. Then the encoded information is sent over the BI-AWGN channel. When transmitting a binary codeword on the BI-AWGN channel with the BPSK assumption, the encoded bits $u_i \in \{0, 1\}$ can be mapped to the symbols $c_i \in \{1, -1\}$ (or $\{0 \to 1, 1 \to -1\}$). As the channel is noisy (Gaussian), errors will be introduced into individual bits. Therefore, corrupted symbols (bits) are be obtained at the receiver side. Before decoding starts, the LLR of each bit (VN) is calculated, which is termed as channel information (LCs). Now, these LCs are fed into the LT decoder. The message passing algorithm is applied over a number of iteration on the Tanner graph of the H-matrix to find out LLR (R) values of the turbo encoded symbols. The detail of the decoding process and calculations for LT decoding over noisy channel is already discussed in Section 3.3. These LLR (R) values of turbo encoded symbols are then fed into turbo decoder as a priori information which are processed by using iterative decoding algorithm (BCJR algorithm) over a number of iteration. The details of turbo decoding is discussed in Section 4.7. The LLR (L) values at the output of the Turbo Decoder are the final

soft information about the source symbols. Based on these final soft information, the final decision about whether the transmitted bit was 0 (if LLR (L) > 0) or 1 (if LLR (L) < 0) is made at the receiver (destination).

5.2 Simulation and Results

Computer simulations were done using C/C++ programming. The data block of size 800 bits was encoded at first by using RCPT codes of rate 4/5 giving a turbo encoded message of block size 1000 bits. This turbo encoded message block of 1000 bits were then further encoded by using LT encoder to get a block of 1300 bits. Therefore, for LT codes, the size of message block is $K = 1000$ and size of encoded message is $N = 1300$. The simulations were done for 1000 different data blocks at different SNR ranging from 0 to 5.5 dB. The BER for both LT and turbo decoder were calculated for each data blocks at all the SNR values. We wanted to see whether the BER that we obtained after LT decoding is reduced or not after turbo decoding. What we have found is that if the number of bits in error after LT decoding is high, turbo decoder can not correct those errors. But, if the number of bits in error after LT decoding is low, turbo decoder is capable of correcting few or all of those errors thus reducing the BER. We have used the scattering graph to present our results.

In all the graphs presented here, the x-axis represents the number of data blocks included in the graph while y-axis represents the BER for both LT and turbo decoding. Only 100 BER values are presented in first three graphs and 1000 BER values are presented in the last one. The " $\bullet$ " represent the BER for LT decoding while " $+$ " represent the BER for turbo decoding. We can see in all the graphs that on the top where the BER is very high, " $\bullet$ " and " $+$ " tends to coincide with each other or are very close to each other indicating that the number of errors after LT decoding are not at all corrected by turbo decoding. As we go down the graph, we can see that " $+$ " starts falling below the " $\bullet$ " which indicates that the errors left after LT decoding are being corrected by turbo decoding to some extent. In Figure 5.3, we can see that there is no " $+$ " sign below many " $\bullet$ " signs. It means

that the errors left after LT decoding in those particular data blocks have been completely corrected by turbo decoding. Observing the graph, we can see that up to 40 out of 1000 bits in errors left after LT decoding can be corrected by turbo decoding. We have seen such pattern throughout all the SNR levels. Particularly, when LT decoding is unsuccessful resulting into few bits in error (around $< 4-5\%$ of total bits), then turbo decoding corrects almost all those errors most of the time. In Figure 5.4, we can see 48 " $\bullet$ " signs. This means that 48 out of 100 data blocks have certain BER indicating that they have some bits in errors after LT decoding. These data blocks with some bits in errors when further processed by turbo decoding, only 4 of them contain some errors as indicated by " $+$ " sign while the errors in all other 44 data blocks are completely corrected. In Figure 5.5, we can still see the similar performances where BER of LT and turbo decoding for 1000 different data blocks are presented. So, we can see that some of the errors left after LT decoding can be corrected by using turbo decoding thus providing the error controlling to the LT codes. The performance is even better as the SNR increases.

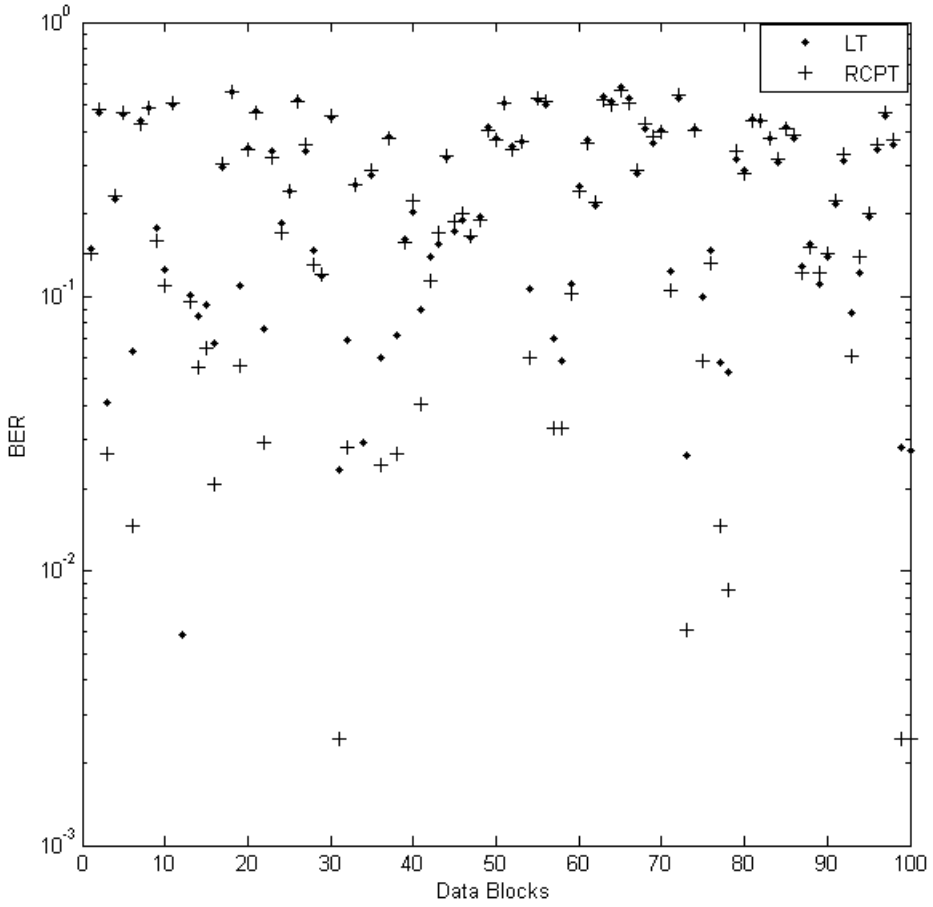


Figure 5.2. Comparison of BER between LT and turbo decoding at SNR of 2 dB.

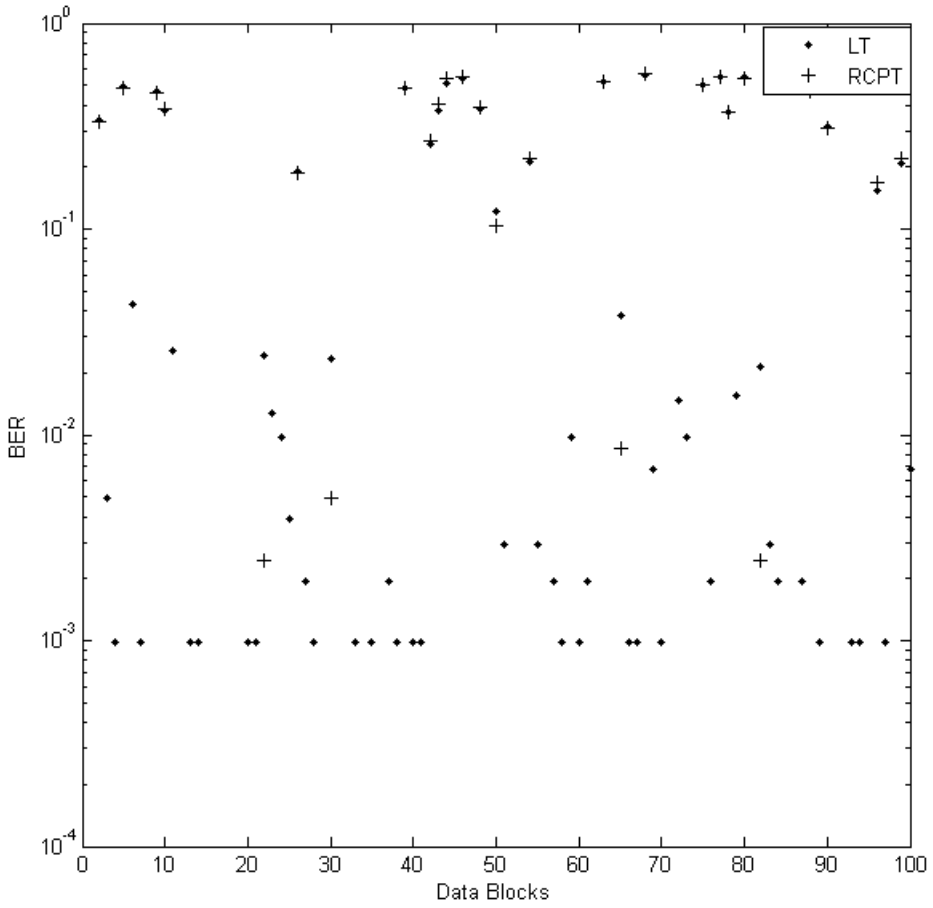


Figure 5.3. Comparison of BER between LT and turbo decoding at SNR of 3 dB.

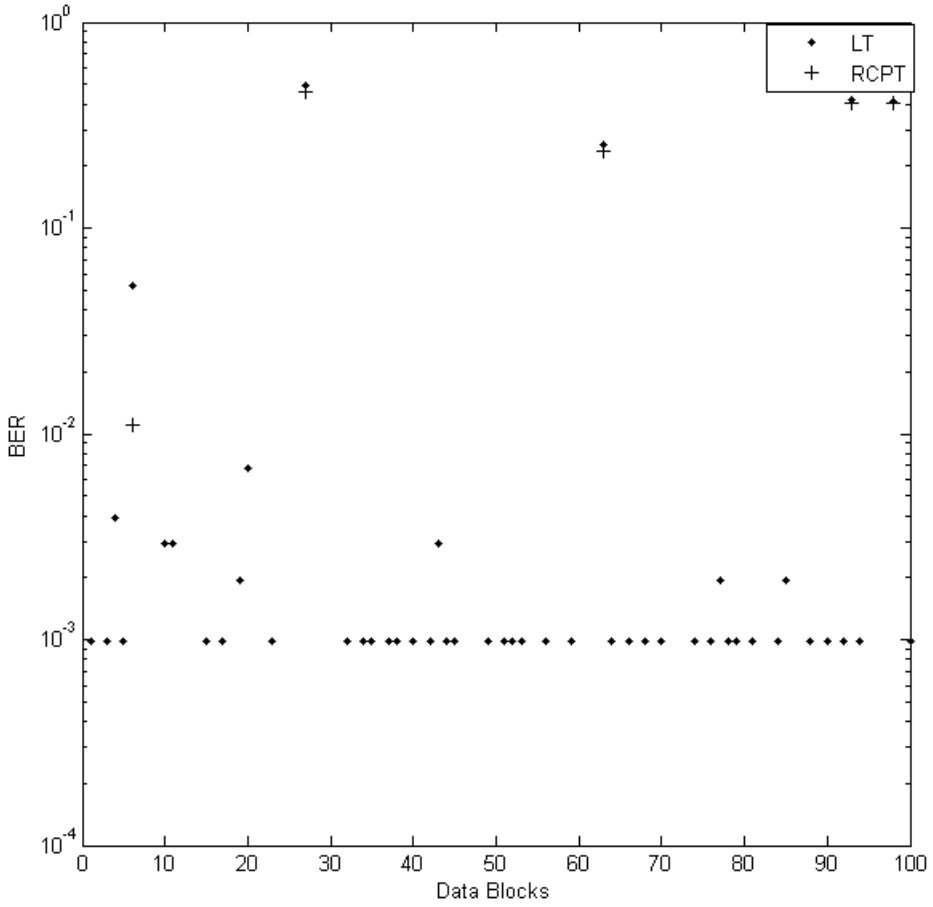


Figure 5.4. Comparison of BER between LT and turbo decoding at SNR of 4 dB.

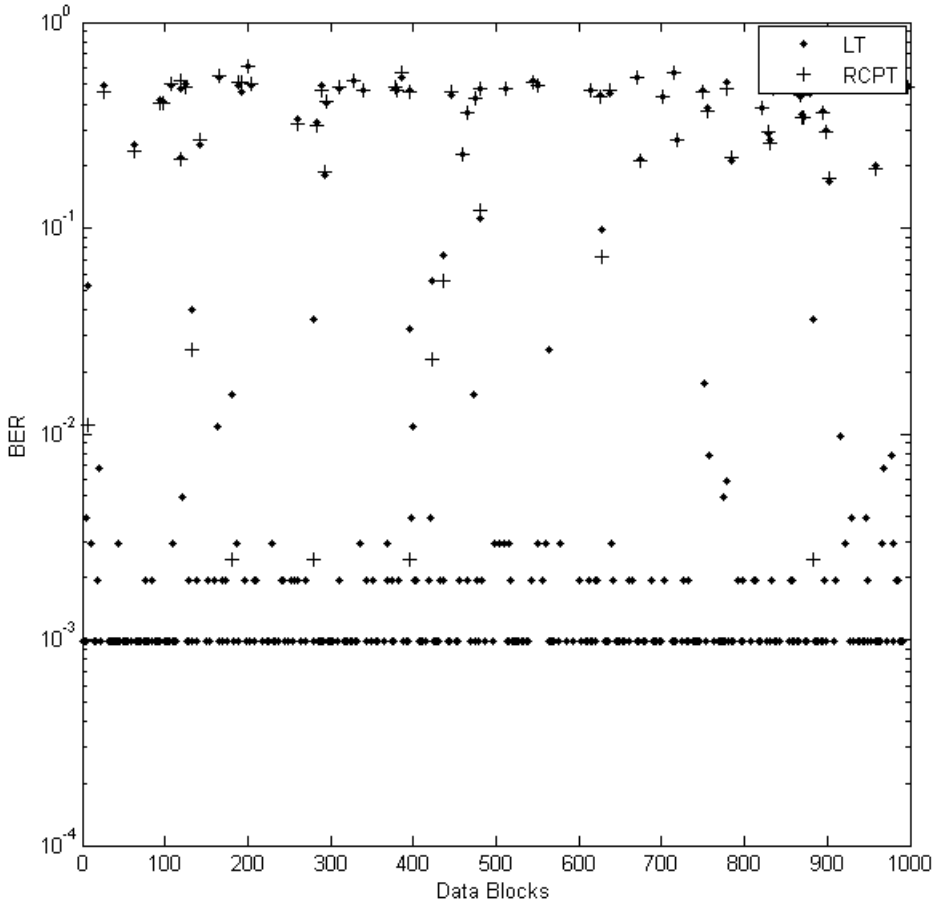


Figure 5.5. Comparison of BER between LT and turbo decoding at SNR of 4 dB. BER for 1000 different data blocks are shown.

Chapter 6

Conclusion and Future Work

6.1 Conclusion

In this thesis, we integrated LT codes and turbo codes for transmission of information over BI-AWGN channel to achieve better overall performance of the system compared to the performance of LT codes alone. To implement the idea, the source data are at first encoded using RCPT codes followed by LT codes and at the receiving end, LT decoding is followed by turbo decoding. Turbo codes, which are good error correcting codes are found to correct some of the errors left after LT decoding. The simulation results are also presented here but the detailed analysis is not done. This concept can be further researched and used to provide better error control property for fountain codes. As the channel considered for the work is AWGN, the performance of LT codes over the channel is also investigated. The algorithm used in the decoding of LDPC codes are used for the decoding of LT codes over AWGN channel. Tanner graphs were used to show the connections between source and encoded packets, where the soft information were propagated along the edges iteratively over a number of iterations. Simulation results show that its performance is not impressive even through the complexity is quite high.

The other objective of the thesis was also met as we provided unequal error protection to two disjoint sources which are communicating to a common destination via a common relay. The RSD is decomposed into two entirely different degree distributions by deconvolution and these distributions are used to encode the information of two sources. One of the degree

distributions is found to provide better protection of the data than the other, which is clearly observed in the simulation results presented.

6.2 Future Work

In the ending section, we present some possible future work based on the theories and results provided in this thesis.

1. We have not achieved good results of LT codes over AWGN channel. There are some possible ways for carrying out further research in this area. The robust soliton distribution and the one we used in our case limit a very small fraction of degree-1 encoded symbols. A suitable degree distribution can be expected.

2. Regarding LT decoding, there might be one or more stopping sets in the generator matrix of the LT code or some particular structures which is forcing for decoding process to fail. If we can find out this, the decoder can further correct more errors and lower the bit error rates. This will also affect the performance when LT codes combined with turbo codes are implemented.

3. The concept of LT codes combined with turbo codes can be implemented in the broadcasting and multicasting scenario to investigate the performances. The work can further be extended to provide unequal error protection to source data when transmitted over noisy channel.

4. In this thesis, LT decoding is followed by turbo decoding. The soft information produced by LT decoder are fed into turbo decoder but the soft information obtained after turbo decoding is not fed back to LT decoding. Therefore, the process of passing the soft information from LT to turbo decoder and from turbo to LT decoder can be performed over a number of iterations to evaluate the performance of the combined codes (LT + turbo).

5. Both LT and turbo codes have their own code rates. Given an overall code rate, tradeoff of code rates between LT and turbo codes can be made and the performances can be compared.

6. We know that the RCPT codes provide layered protection. Further research can be carried out to study whether the layered protection can be provided when LT codes are combined with RCPT codes.

Bibliography

Bibliography

[1] S. Puducheri, J. Kliewer, and T.E. Fuja. The Design and Performance of Distributed LT Codes. *IEEE Transactions on Information Theory*, 53(10):3740–3754, Oct. 2007.

[2] G. Solomon and I.S. Reed. Polynomial codes over certain finite fields. *Journal of the Society for Industrial and Applied Mathematics*, 8(2):300–304, June 1960.

[3] D. J C MacKay. Fountain Codes. *IEE Proceedings on Communications*, 152(6):1062–1068, 2005.

[4] J.W. Byers, M. Luby, and M. Mitzenmacher. Accessing multiple mirror sites in parallel: using Tornado codes to speed up downloads. In *INFOCOM '99. Eighteenth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*, volume 1, pages 275–283 vol.1, 1999.

[5] Michael Luby. LT Codes. In *Proceedings of the 43rd Symposium on Foundations of Computer Science*, FOCS '02, pages 271–280, Washington, DC, USA, 2002. IEEE Computer Society.

[6] M. Mitzenmacher. Digital fountains: a survey and look forward. In *Information Theory Workshop, 2004. IEEE*, pages 271–276, 2004.

[7] P. Elias. Coding for two noisy channels. *Information Theory, Third London Symposium, Butterswot's Scientific Publications*, pages 61–76, 1955.

[8] S. Puducheri, J. Kliewer, and T.E. Fuja. Distributed LT Codes. In *IEEE International Symposium on Information Theory*, pages 987–991, 2006.

[9] J.W. Byers, M. Luby, and M. Mitzenmacher. A digital fountain approach to asynchronous reliable multicast. *IEEE Journal on Selected Areas in Communications*, 20(8):1528–1540, 2002.

[10] T. Sikora. MPEG digital video-coding standards. *Signal Processing Magazine, IEEE*, 14(5):82–100, 1997.

[11] B. Masnick and J. Wolf. On linear unequal error protection codes. *IEEE Transactions on Information Theory*, 13(4):600–607, 1967.

[12] N. Rahnavard, B.N. Vellambi, and F. Fekri. Rateless Codes With Unequal Error Protection Property. *IEEE Transactions on Information Theory*, 53(4):1521–1532, 2007.

[13] A. Shokrollahi. Raptor codes. *IEEE Transactions on Information Theory*, 52(6):2551–2567, 2006.

[14] R. Palanki and Jonathan S. Yedidia. Rateless codes on noisy channels. In *International Symposium on Information Theory, ISIT 2004*, 2004.

[15] O. Etesami and A. Shokrollahi. Raptor codes on binary memoryless symmetric channels. *IEEE Transactions on Information Theory*, 52(5):2033–2051, 2006.

[16] R. Gallager. Low-density parity-check codes. *IEEE Transactions on Information Theory*, pages 21–28, Jan. 1962.

[17] D. J C MacKay. Good error-correcting codes based on very sparse matrices. *IEEE Transactions on Information Theory*, 45(2):399–431, 1999.

[18] N. Alon and M. Luby. A linear time erasure-resilient code with nearly optimal recovery. *IEEE Transactions on Information Theory*, 42(6):1732–1736, 1996.

[19] R.M Tanner. A recursive approach to low complexity codes,. *IEEE Transactions on Information Theory*, pages 533–547, Sept. 1981.

[20] William E. Ryan and Shu Lin. *Channel Codes: Classical and Modern*. Cambridge University Press, Sept. 2009.

[21] C.E. Shannon. Communication in the Presence of Noise. *Proceedings of the IRE*, 37(1):10–21, 1949.

[22] C. Berrou, A. Glavieux, and P. Thitimajshima. Near Shannon limit error-correcting coding and decoding: Turbo-codes. 1. In *IEEE International Conference on Communications, 1993. ICC '93 Geneva. Technical Program, Conference Record*, volume 2, pages 1064–1070 vol.2, 1993.

[23] P. Elias. Coding for noisy channels. *IRE Conv. Rep., Pt. 4*, 37:37–47, 1955.

[24] C. Berrou and A. Glavieux. Near optimum error correcting coding and decoding: turbo-codes. *IEEE Transactions on Communication*, 44(10):1261–1271, 1996.

[25] S. Adrian Barbulescu and Steven S. Pietrobon. TURBO CODES: a tutorial on a new class of powerful error correcting coding schemes - Part i: Code Structures and Interleaver Design, 1998.

[26] L. Bahl, J. Cocke, F. Jelinek, and J. Raviv. Optimal decoding of linear codes for minimizing symbol error rate (Corresp.). *IEEE Transactions on Information Theory*, 20(2):284–287, 1974.

[27] J.P. Woodard and L. Hanzo. Comparative study of turbo decoding techniques: an overview. *IEEE Transactions on Vehicular Technology*, 49(6):2208–2233, 2000.

Vita

Amrit Kharel was born on June 21, 1984 in Nepal. He obtained a Bachelor of Engineering degree in Electronics and Communication Engineering from Pulchowk Campus, Institute of Engineering, Tribhuvan University, Nepal in year 2009.

In January 2011, he joined the University of Mississippi for a M.S. degree with emphasis in Telecommunications. While pursuing his master's degree he worked as a Research Assistant and Lab Instructor at the department of Electrical Engineering. He received his master's degree in August, 2013.