POINT TO MULTIPOINT DISTRIBUTION USING FOUNTAIN CODES

Project Report submitted in partial fulfillment of the requirement for the degree of

Bachelor of Technology

in

Electronics and Communication Engineering

Under the Supervision of

Dr. Mohammed Usman

By

Saksham Kaushik (091013)

Rajul Agarwal (091073)

Abhay Verma (091095)

Jaypee University of Information and Technology

Waknaghat, Solan – 173234, Himachal Pradesh

CERTIFICATE

This is to certify that project report entitled "**Point to Multipoint Distribution using Fountain Codes**", submitted by "**Saksham Kaushik, Rajul Agarwal and Abhay Verma**" in partial fulfillment for the award of degree of Bachelor of Technology in Electronics and Communication Engineering to Jaypee University of Information Technology, Waknaghat, Solan has been carried out under my supervision.

This work has not been submitted partially or fully to any other University or Institute for the award of this or any other degree or diploma.

[illegible]

Date: 24th May 2013 **Dr.Mohammed Usman**

Assistant Professor

Department of Electronics & Communication

ACKNOWLEDGEMENT

Firstly, we would like to thanks God for his grace and our parents for their constant motivation that has helped us in the completion of this project.

We would like to express our deep sense of gratitude to Dr. Mohd Usman who provided us with this opportunity of undertaking this project under his able guidance. His immense knowledge and unparalleled support has motivated us all through to accomplish the completion of this project.

We would also like to extend our thanks to Dr. Sunil Bhooshan(Head of Department, ECE) and Dr.T.S Lamba (Dean) who have been supportive throughout the work of this project. The zeal to accomplish this project could not have been realized without the support of the entire Electronics & Communication faculty and support staff.

Date – 24th May 2013	**Saksham Kaushik**	**(091013)**	Saksham Kaushik
	Rajul Agarwal	**(091073)**	Rajul Agarwal
	Abhay Verma	**(091095)**	[illegible]

Table of Content

List of Figures

Abbreviations and Symbols

1. TCP - Transmission Control Protocol
2. LT codes – Luby Transform Codes
3. FEC – Forward error correction
4. AWGN – Additive White Gaussian Noise
5. BEC- Binary Erasure channel
6. Ack – Acknowledgement
7. RS Codes- Reed Solomon Codes
8. LDPC – Low Density Parity Check Code
9. HDPC – High Density Parity Check Code
10. 'n'- No. of message bits
11. 'k'- No. of encoded bits
12. δ -Allowable Failure Probability
13. d – Degree of the encoded symbols
14. p(i) – Ideal Soliton Distribution
15. O – Big-O Notation

Abstract

This project addresses reliable file delivery over broadcast networks, concentrating on the Digital Fountain codes.

Firstly, describing Luby-Transform (LT) codes, which are the first practical Fountain codes wherein Fountain codes are basically a class of the Erasure codes or forward error correction codes (FEC). Then, using a natural and easy to understand linear algebra notation, we describe Fountain codes as a powerful extension of LT codes.

Thus in the subsequent chapters we have tried to implement an approach where packets are not ordered and the recovery of some subset of packets will allow for successful decoding. This class of codes, called Fountain codes, was pioneered by a start-up called Digital Fountain and has greatly influenced the design of codes for binary erasure channels (BECs), a well-established model for the Internet.

We provide some insight into the Fountain code structure and some guidelines for implementation of encoders and decoders. Through this approach, we would like to implement data transmission using Fountain Codes or Luby transform codes that are practically realized by using a distribution function known as the Robust Soliton Distribution. These Fountain codes are further used to realize practical point to multipoint distribution allowing broadcast of information over noisy channels, without requiring retransmissions. The addition of noise (AWGN) by the erasure channel further requires us to modify certain characteristics in the coding specifications thus allowing for successful decoding of the encoded data.

CHAPTER I : Introduction

In this chapter we shall discuss how unicast based protocols like TCP are used for transmission of data and what are the problems encountered at the receiver end when these are used. We shall also elaborate upon the need for error correction and detection and how Erasure codes are better in this regard.

Consider a setting where a large file is disseminated to a wide audience who may want to access it at various times and have transmission links of different quality. Normally when data is to be transmitted on an IP based network, it is partitioned into equal length pieces and placed into packets. Current networks use unicast-based protocols such as the transport control protocol (TCP), which requires a transmitter to continually send the same packet until acknowledged by the receiver. TCP establishes a connection between the sender and the receiver. The sender sends a packet, then waits for acknowledgement from the receiver before sending another. If a packet goes too long without being acknowledged, it resends the packet until it does receive an acknowledgement since that is how it gets its reliability. This kind of architecture therefore places a strong importance on the requirement of a reverse channel, wherein acknowledgments are sent upon the successful reception of these ordered packets at the receiver.

1.1 Problems with the Traditional Protocols

Reliable transmission of data over the wireless links has been the subject of much research. For the most part, reliability is guaranteed by use of appropriate protocols. For example, the ubiquitous TCP/IP ensures reliability by essentially retransmitting packets within a transmission window whose reception has not been acknowledged by the receiver (or packets for which the receiver has explicitly sent a negative acknowledgment). It is well known that such protocols exhibit poor behavior in many cases, such as transmission of data from one server to multiple receivers, or transmission of data over heavily impaired channels, such as poor wireless or satellite links. Moreover, ack-based protocols such as TCP perform poorly when the distance between the sender and the receiver is long, since large distances lead to idle times during which the sender waits for an acknowledgment and cannot send data.

In the case of multicast and broadcast, the sender doesn't even know how many receivers there might be, not to mention who they are. That makes it pretty much impossible for it to wait for confirmation and re-send packets if some receiver doesn't acknowledge a packet correctly. The process involving re-transmission of these packets leads to bandwidth wastage since a lot of these packets that do not reach the receiver are required to be sent again until an acknowledgement is received.

It can easily be seen that this architecture does not scale well when many users access a server concurrently and is extremely inefficient when the information transmitted is always the same. In effect, TCP and other unicast protocols place strong importance on the *ordering of packets* to simplify coding at the expense of increased traffic.

Further, TCP/IP does not scale well to transmission over networks with high latencies, to transmit over networks with high loss rates, or to highly concurrent transmission of the same content to multiple clients. For many applications TCP is not appropriate. One problem with normal implementations is that the application cannot access the packets coming after a lost packet until the retransmitted copy of the lost packet is received. This causes problems for real-time applications such as streaming media, real-time multiplayer games and voice over IP (VoIP) where it is generally more useful to get most of the data in a timely fashion than it is to get all of the data in order. *Erasure codes* have been proposed to provide reliability for a variety of data delivery applications.

For the one-to-many data delivery problem, where scalability of the sender and the network is of paramount importance, feedback to the sender needs to be limited and sending packets that are redundant for some receivers is wasteful, reliability may be able to be provided using *Erasure codes* . The attractiveness of this approach is that if the erasure codes are powerful enough then a single sender can potentially be used to reliably deliver data efficiently to a large number of concurrent receivers without feedback. Erasure codes can be used in conjunction with these congestion control protocols to help provide a complete reliable delivery transport.

The one-to-many problem is even more difficult when different receivers have different bandwidth connections to the sender and it is desired that each receiver obtain the data at the fastest possible speed independent of other receivers. Receiver-driven congestion control protocols have been designed independently of a reliability protocol that scalably delivers a different fraction of the packets generated to each receiver depending on current network conditions and the receivers bandwidth connection to the sender.
Erasure codes can be used in conjunction with these congestion control protocols to help provide a complete reliable delivery transport.

For many-to-one delivery it is useful for receivers to be able to concurrently receive packets for the same data from multiple senders potentially at different locations. In this case receiving redundant packets is a concern, and erasure codes can be used to minimize delivery of redundant packets. *LT codes* offers compelling advantages for all these different types of data delivery applications by minimizing the number of message symbols that are required to perform the underlying process.There are a variety of other applications of LT codes, including robust distributed storage, delivery of streaming content, delivery of content to mobile clients in wireless networks, delivery of content along multiple paths to ensure resiliency to network disruption.

For all the aforesaid reasons, other transmission solutions have been proposed.
One class of such solutions is based on coding. The original data is encoded using some linear erasure correcting code. If during the transmission some part of the data is lost, then it is possible to recover the lost data using erasure correcting algorithms.
For applications it is crucial that the codes used are capable of correcting as many erasures as possible, and it is also crucial that the encoding and decoding algorithms for these codes are very fast.

1.2 Error Correction Codes

As detailed earlier Erasure codes are a proposal to provide reliability for a variety of data delivery applications and also to solve certain other problems that have been encountered while using traditional protocols like TCP. Many communication channels are subject to channel noise, and thus errors may be introduced during transmission from the source to a receiver. Error detection techniques allow detecting such errors, while error correction enables reconstruction of the original data. Since a transmission channel can introduce errors due to noise, TCP allows for error correction but using Forward error correction ***(FEC)*** codes helps us to ensure reliable delivery of data over unreliable communication channels.

Timeouts and retransmissions handle error control in TCP. Although delay could be substantial, particularly if you were to implement real-time applications, the use of both these techniques offer error detection and error correction thereby guaranteeing that data will eventually be sent successfully.

In telecommunication, information theory, and coding theory, *forward error correction* (FEC) or *channel coding* is a technique used for controlling errors in data transmission over unreliable or noisy communication channels. The central idea is the sender encodes their message in a **redundant** way by using an *error-correcting code* (ECC).

Redundancy in information theory, is the number of bits used to transmit a message minus the number of bits of actual information in the message. The Hamming (7,4) codes was the first of these class of codes.

1.3 Advantage of FEC's over TCP error control

We have described briefly how TCP allows for error control. However when a single sender is sending the same data concurrently to multiple receivers, FEC is more useful since it allows for error correction without the requirement for retransmissions. The redundancy[1] allows the receiver to detect a limited number of errors that may occur anywhere in the message, and often to correct these errors without retransmission. FEC gives the receiver the ability to correct errors without needing a reverse channel to request retransmission of data, but at the cost of a fixed, higher forward channel bandwidth.

FEC is therefore applied in situations where retransmissions are costly or impossible, such as one-way communication links and when transmitting to *multiple receivers in multicast*. FEC's may further be divided into two broad categories: Block Codes and Convolutional Codes, of which the LDPC codes and Turbo Codes are highly efficient linear Block codes that can provide performance very close to channel capacity. In the following chapters we shall see that how these FEC helped us to develop LT codes that we have used for broadcasting.

CHAPTER II : Fountain Codes

In the previous chapter we have described the performance drawbacks of the traditional protocols and the use of FEC'S for error correction. In this chapter, We introduce fountain codes further describing upon LT codes, the first rateless erasure codes that are very efficient as the data length grows.

2.1 Fountain Codes

Fountain codes are ideally suited for transmitting information over computer networks. A server sending data to many recipients can implement a Fountain code for a given piece of data to generate a potentially infinite stream of packets. As soon as a receiver requests data, the packets are copied and forwarded to the recipient. In a broadcast transmission model there is no need for copying the data since any outgoing packet is received by all the receivers. In other types of networks, the copying can be done actively by the sender, or it can be done by the network, for example if multicast is enabled. The recipient collects the output symbols, and leaves the transmission as soon as it has received 'k' of them. At that time it uses the decoding algorithm to recover the original 'n' symbols.
More loss of symbols just translates to a longer waiting time to receive the packets. If 'k' can be chosen to be arbitrarily close to 'n' , then the corresponding Fountain code has a universality property in the sense that it operates close to capacity for *any* erasure channel.

Fountain codes have also other very desirable properties. For example, since each output symbol is generated independently of any other one, a receiver may collect output symbols generated from the same set of input symbols, but by different devices operating a Fountain encoder. This allows for the design of massively scalable and fault-tolerant communication systems over packet-based networks. In order to make Fountain codes work in practice, one needs to ensure that they possess a fast encoder and decoder, and that the decoder is capable of recovering the original symbols from any set of output symbols whose size is close to optimal with high probability. We call such Fountain codes *universal*. The first class of such universal Fountain codes was invented by Michael Luby. The codes in this class are called LT-codes.

LT Codes

It is a forward error correction (FEC) code for the transmission channel, which transforms a message of n symbols into a longer message (code word) with k symbols such that the original message can be recovered from a subset of the k symbols.

2.2 Digital Fountain

The digital fountain was devised as the ideal protocol for transmission of a single file to many users (multicast) who may have different access times and channel fidelity. The name is drawn from an analogy to water fountains, where many can fill their cups with water at any time. The output packets of digital fountains must be universal like drops of water and hence be useful independent of time or the state of a user's channel.

Consider a file that can be split into 'n' packets or information symbols and must be encoded for a transmission channel.
A digital fountain that transmits this file should have the following properties:
1. It can generate an endless supply of encoding packets.
2. A user can reconstruct the file using any k packets with decoding being linear with respect to 'n'.
3. The space needed to store any data during encoding and decoding is linear in 'n'.

These properties show digital fountains are as reliable and efficient as TCP systems, but also universal and tolerant, properties desired in networks.

2.2.1 LT Codes

Luby Transform codes are a class of erasure codes that are rateless, i.e., the number of encoding symbols that can be generated from the data is potentially limitless. Furthermore, encoding symbols can be generated on the fly, as few or as many as needed. Also, the decoder can recover an exact copy of the data from any set of the generated encoding symbols that in aggregate are only slightly longer in length than the data. LT codes are efficient in the sense that the transmitter does not require an acknowledgement (ACK) from the receiver. This property is especially desired in multicast channels because it will significantly decrease the overhead incurred by processing the ACK's from multiple receivers.

Thus, no matter what the loss model is on the erasure channel, 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. Since the decoder can recover the data from nearly the minimal number of encoding symbols possible, this implies that LT codes are near optimal with respect to any erasure channel.

Furthermore, the encoding and decoding times are asymptotically very efficient as a function of the data length. Thus, LT codes are universal in the sense that they are simultaneously near optimal for every erasure channel and they are very efficient as the data length grows.

2.3 Comparison to some other Erasure codes

As mentioned earlier there are a number of erasure codes other than the LT codes. Some of these codes are modifications to the LT codes so developed. However these and some traditional fountain codes are common in the sense that they are all FEC codes. Traditional erasure codes are typically block codes with a fixed rate, i.e., n input symbols are used to generate k-n redundant symbols for a total of k encoding symbols, and the rate of the code is *n/k*. For example, research in networking has suggested using implementations of both Reed-Solomon and Tornado codes for reliable data distribution applications, and in these cases both *k* and *n* either are limited to fairly small values due to practical considerations or are fixed before the encoding process begins.

2.3a) <u>RS Codes</u>

Reed-Solomon (RS) codes are the first example of fountain-like codes because a message of 'n' symbols can be recovered from any 'k' encoding symbols. However, RS codes require quadratic decoding time and are limited to a smaller block length n. Low-density parity-check (LDPC) codes reduce the decoding complexity by use of the sum-product algorithm and iterative decoding techniques. They come closer to the fountain code ideal. However early LDPC codes are restricted to fixed-degree regular graphs due to which significantly more than k encoding symbols are needed to successfully decode the transmitted signal.

2.3b) <u>Tornado Codes</u>

We begin our survey of fountain codes by studying Tornado codes. These codes , were the first steps towards achieving the digital fountain ideal described in Section 2.1. Tornado codes are block codes and hence not rateless. However, they do approach Shannon capacity with linear decoding complexity.

Tornado codes are block erasure codes that have linear in 'k' encoding and decoding times. LT codes are somewhat similar to Tornado codes. For example, they use a similar rule to recover the data, and the degree distributions used for Tornado codes is superficially similar to the degree distributions used for LT codes. However, the actual degree distribution used for Tornado codes turns out to be inapplicable to LT codes. However, LT codes have significant application advantages over Tornado codes. Let $c = k/n$ be the constant stretch factor of the Tornado code design. Once *k* and *n* have been fixed the Tornado encoder produces *k* encoding symbols, and cannot produce further

encoding symbols on the fly if the demand arises. In contrast, the encoder can generate as few or as many encoding symbols as needed on demand.
We consider a system model in which a single transmitter performs bulk data transfer to a larger number of users on an erasure channel. Our objective is to achieve complete file transfer with the minimum number of encoding symbols and low decoding complexity. For 'n' information symbols, RS codes can achieve this with k log k encoding and quadratic decoding times. The reason for the longer decoding time is that in RS codes, every redundant symbol depends on all information symbols. By contrast, every redundant symbol depends only on a small number of information symbols in Tornado codes. Thus they achieve linear encoding and decoding complexity, with the cost that the user requires slightly more than k packets to successfully decode the transmitted symbols.

Both Reed-Solomon and Tornado codes are *systematic codes,* i.e. all the input symbols that constitute the data are directly included among the encoding symbols, whereas LT codes are *not systematic.*

2.3b).1 <u>*Disadvantages of Traditional Block Codes*</u>

There are more disadvantages of traditional block codes when it comes to their use for data transmission. The model of a single erasure channel is not adequate for cases where data is to be sent concurrently from one sender to many receivers. In this case, the erasure channels from the sender to each of the receivers have potentially different erasure probabilities. Typically in applications, the sender or the receiver may probe their channels so the sender has a reasonable guess of the current erasure probability of the channel and can adjust the coding rate accordingly. But if the number of receivers is large, or in situations such as satellite or wireless transmission where receivers experience sudden abrupt changes in their reception characteristics, it becomes unrealistic to assume and keep track of the loss rates of individual receivers.

The sender is then forced to assume a worst case loss rate for all the receivers. This not only puts unnecessary burdens on the network if the actual loss rate is smaller, but also compromises reliable transmission if the actual loss rate is larger than the one provisioned for.

2.3c) Raptor Codes

Raptor codes are an enhancement to the LT codes and these are the fountain codes with linear time encoding and decoding. Although the basic concept behind a fountain code is realized in these, the idea behind Raptor codes is a pre-coding of the input symbols prior to the application of an appropriate LT-code.These modifications have helped us achieve linear encoding and decoding that is required in specific applications, such that they are advantageous in comparison to the LT codes.

LT codes illuminated the benefits of considering rateless codes in realizing a digital fountain. However, they require the decoding cost to be O(ln n) in order for every information symbol to be recovered and decoding to be successful. Raptor (rapid Tornado) codes were developed as a way to reduce decoding cost to O(1) by preprocessing the LT code with a standard erasure block code (such as a Tornado code).

Raptor codes are formed by the concatenation of two codes.
A fixed rate erasure code, usually with a fairly high rate, is applied as a 'pre-code' or 'outer code'. This pre-code may itself be a concatenation of multiple codes, for example LDPC concatenated with HDPC. The inner code takes the result of the pre-coding operation and generates a sequence of encoding symbols. The inner code is a form of LT codes. Each encoding symbol is the XOR of a randomly chosen set of symbols from the pre-code output.

This distribution, as well as the mechanism for generating random numbers for sampling this distribution and for choosing the symbols to be XOR'ed, must be known to both sender and receiver. In the case of non-systematic raptor codes, the source data to be encoded is used as the input to the pre-coding stage. In the case of systematic raptor codes, the input to the pre-coding stage is obtained by first applying the inverse of the encoding operation that generates the first k output symbols to the source data.

One of the disadvantages of LT- or Raptor codes is that they are not systematic. This means that the input symbols are not necessarily reproduced among the output symbols. The straightforward idea of transmitting the input symbols prior to the output symbols produced by the coding system is easily seen to be flawed, since this does not guarantee a high probability of decodability from any subset of received output symbols.

2.4 LT Codes Design

Having described in detail the fundamentals of fountain codes and a few classes of the forward error correction codes, we shall now discuss in detail the Luby Transform codes and how their practical implementations may be realized further using them for multicasting of information.

The key to the design and analysis of the LT codes is the introduction and analysis of the LT process. This process and its analysis is a novel generalization of the classical process and analysis of throwing balls randomly into bins and it may be of independent interest. We provide a full analysis of the behavior of the LT process using first principles of probability theory, which precisely captures the behavior of the data recovery process. The symbol length for the codes can be arbitrary, from one-bit binary symbols to general k bit symbols. We analyze the run time of the encoder and decoder in terms of symbol operations, where a symbol operation is either an exclusive-or of one symbol into another or a copy of one symbol to another. If the original data consists of *'n'* input symbols then each encoding symbol can be generated, independently of all other encoding symbols, on average by $O(\ln(n/\delta))$ symbol operations, and the *'n'* original input symbols can be recovered from any $n+O(\sqrt{n}\ \ln2(n/\delta))$ of the encoding symbols with probability $1-\delta$ by on average $O(n.\ \ln(n/\delta))$ symbol operations.

The length *k* of the encoding symbols can be chosen as desired. The overall encoding and decoding is more efficient in practice for larger values of *k* because of overheads with bookkeeping operations, but the value of *k* has no bearing on the theory. In transport applications *k* is sometimes chosen to be close to the length of the packet payload.

2.5 Code Construction

In the analysis of the LT process, encoding symbols are analogous to balls and input symbols are analogous to bins, and the process succeeds if at the end all input symbols are covered. All input symbols are initially uncovered. At the first step all encoding symbols with one neighbor are released to cover their unique neighbor. The set of covered input symbols that have not yet been processed is called the ripple, and thus at this point all covered input symbols are in the ripple.

At each subsequent step one input symbol in the ripple is processed: it is removed as a neighbor from all encoding symbols which have it as a neighbor and all such encoding symbols that subsequently have exactly one remaining neighbor are released to cover their remaining neighbor. Some of these neighbors may have been previously uncovered, causing the ripple to grow, while others of these neighbors may have already been in the ripple, causing no growth in the ripple. The process ends when the ripple is empty at the end of some step. The process fails if there is at least one uncovered input symbol at the end. The process succeeds if all input symbols are covered by the end.

Balls And Bins : An Analogy

In the classical balls and bins process, all the balls are thrown at once. Because of the high probability of collisions, i.e., multiple balls covering the same bin, many more balls must be thrown to cover all bins than there are bins. In contrast, the proper design of the degree distribution ensures that the LT process releases encoding symbols incrementally to cover the input symbols.

Initially only a small fraction of the encoding symbols are of degree one and thus they cover only a small fraction of the input symbols. Covered input symbols are processed one at a time, and each one processed releases potentially other encoding symbols to randomly cover unprocessed input symbols. The goal of the degree distribution design is to slowly release encoding symbols as the process evolves to keep the ripple small so as to prevent redundant coverage of input symbols in the ripple by multiple encoding symbols, but at the same time to release the encoding symbols fast enough so that the ripple does not disappear before the process ends.

Because the neighbors of an encoding symbol are chosen at random independently of all other encoding symbols, it is easy to verify that the step at which a particular encoding symbol is released is independent of all the other encoding symbols. Furthermore, once an encoding symbol is released it covers a uniformly chosen input symbol among the unprocessed input symbols, independent of all other encoding symbols. These properties make the LT process especially amenable to the design of good degree distributions and analysis of the process with respect to these distributions.

It is not hard to verify that there is a one to one correspondence between the LT process and the decoder, i.e., an encoding symbol covers an input symbol if and only if the encoding symbol can recover that input symbol. Thus, the LT process succeeds if and only if the decoder successfully recovers all symbols of the data. The total number of encoding symbols needed to cover all the input symbols corresponds exactly to the total number of encoding symbols needed to recover the data. The sum of the degrees of these encoding symbols corresponds exactly to the total number of symbol operations needed to recover the data.

2.5.1 Encoding Details

The encoder works as follows-
Given an ensemble of encoding symbols and some representation of their associated degrees and sets of neighbors, the decoder repeatedly recovers input symbols using the following rule as long as it applies.

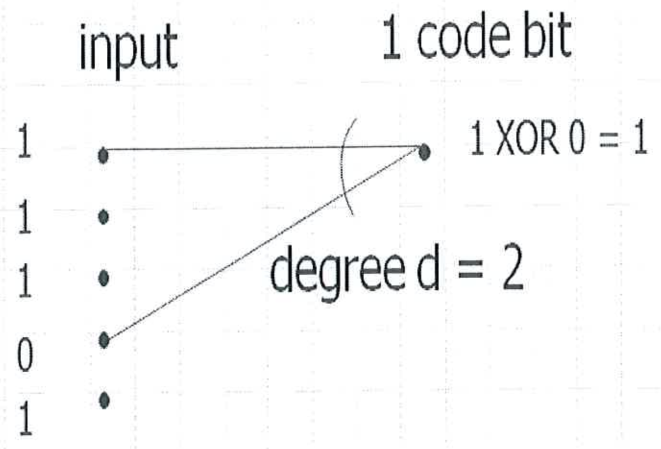
When using the encoding symbols to recover the original input symbols of the data, the decoder needs to know the degree and set of neighbors of each encoding symbol. There are many different ways of communicating this information to the decoder, depending on the application. For example, the degree and a list of neighbor indices may be given explicitly to the decoder for each encoding symbol. As another example, the degree and neighboring indices of each encoding symbol may be computed by the decoder implicitly based for example on the timing of the reception of the encoding symbol or the position of the encoding symbol relative to the positions of other encoding symbols. As another example, a key may be associated with each encoding symbol and then both the encoder and decoder apply the same function to the key to produce the degree and set of neighbors of the encoding symbol. In this case, the encoder may randomly choose each key it uses to generate an encoding symbol and keys may be passed to the decoder along with the encoding symbols. There are a variety of ways to associate a degree and a set of neighbors with an encoding symbol depending on the application.

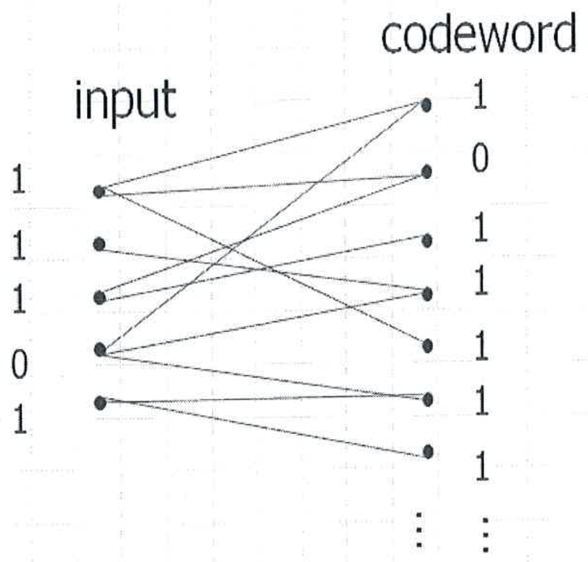
2.5.2) Encoding Process

Any number of encoding symbols can be independently generated from k information symbols by the following encoding process:

- Determine the degree d of an encoding symbol. The degree is chosen at random from a given node degree distribution P(x) which shall be the focus of the remainder of this chapter.
- Choose d distinct information symbols uniformly at random. They will be neighbors of the encoding symbol.
- Assign the XOR of the chosen d information symbols to the encoding symbol.

This process is similar to generating parity bits except that only the parity bits are transmitted. The degree distribution P(x) comes from the sense that we can draw a bipartite graph, such as in Figure , which consists of information symbols as variable nodes and encoding symbols as factor nodes. The degree distribution determines the performance of LT codes, such as the number of encoding symbols and probability of successful decoding.





Generator Matrix (Gm[n,k])

This is a n x k matrix where 1's in corresponding columns depict respective message bit, from which encoded bit is obtained. The shaded columns represent lost encoded symbols.

Encoded bit (k) ->

Message bit (N) ->

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

2.5.3 <u>Decoding Process</u>

The encoding symbols are transmitted through a transmission or an erasure channel with the probability of erasure p. The special characteristic of an erasure channel is that receivers have correct data or no data. There is no confusion where the decoder needs to "guess" the original data; it recovers the true data or gives up. For decoding of LT codes, a decoder needs to know the neighbors of each encoding symbol.

This information can be transferred in several ways. For example, a transmitter can send a packet, which consists of an encoding symbol and the list of its neighbors. An alternative method is that the encoder and the decoder share a random number generator seed, and the decoder finds out the neighbors of each encoding symbol by generating random linear combinations synchronized with the encoder.

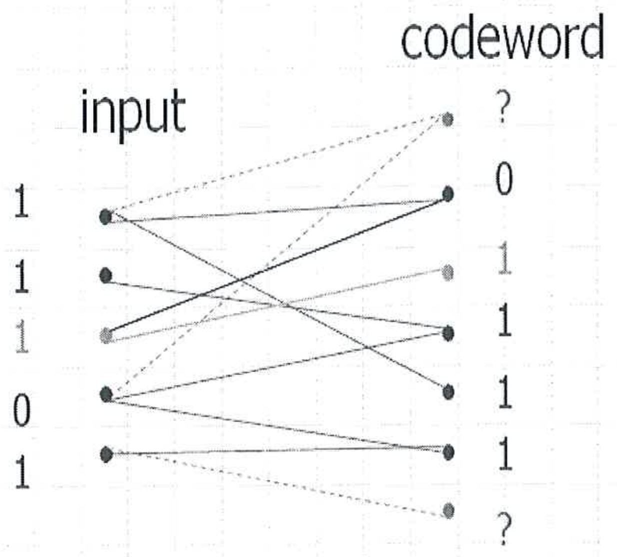
The decoder recovery rule:

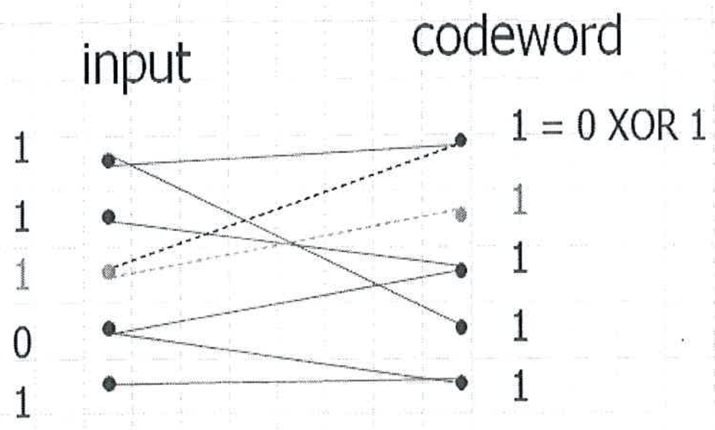
If there is at least one encoding symbol that has exactly one neighbor then the neighbor can be recovered immediately since it is a copy of the encoding symbol. The value of the recovered input symbol is exclusive-ored into any remaining encoding symbols that also have that input symbol as a neighbor, the recovered input symbol is removed as a neighbor from each of these encoding symbols and the degree of each such encoding symbol is decreased by one to reflect this removal.

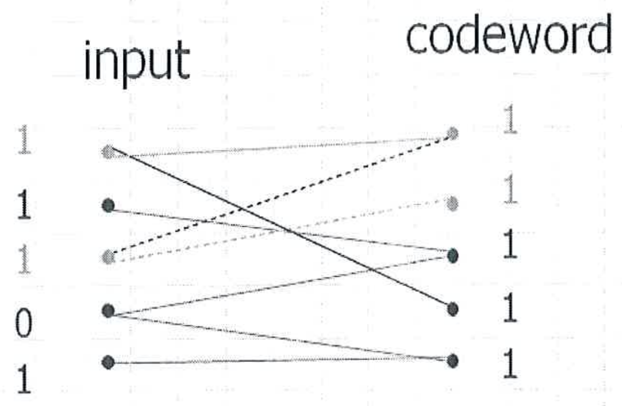
With the encoding symbols and the indices of their neighbors, the decoder can recover information symbols with the following three-step process, which is called LT process:

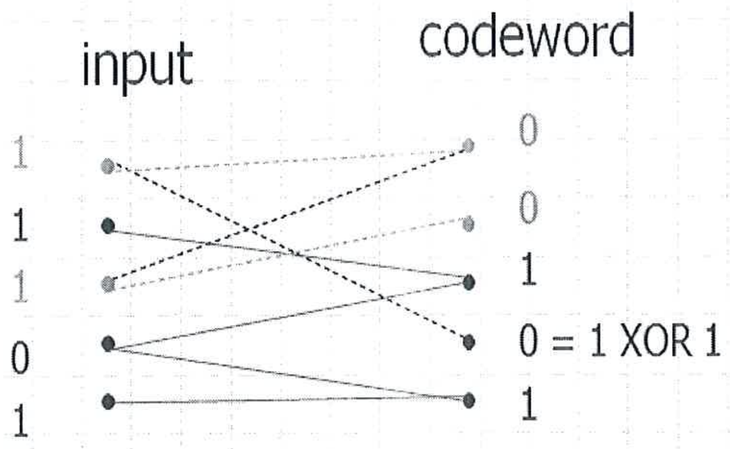
- All encoding symbols of degree one, i.e., those which are connected to one information symbol, are released to cover their unique neighbor.
- The released encoding symbols cover their unique neighbor information symbols. In this step, the covered but not processed input symbols are sent to ripple, which is a set of covered unprocessed information symbols gathered through the previous iterations.
- One information symbol in the ripple is chosen to be processed: the edges connecting the information symbol to its neighbor encoding symbols are removed and the value of each encoding symbol changes according to the information symbol. The processed information symbol is removed from the ripple.

The decoding process continues by iterating the above three steps. Since only encoding symbols of degree one can trigger each iteration, it is important to guarantee that there always exist encoding symbols of degree one to release during the process for successful recovery. If there is at least one encoding symbol that has exactly one neighbor then the neighbor can be recovered immediately since it is a copy of the encoding symbol. The value of the recovered input symbol is exclusive-ored into any remaining encoding symbols that also have that input symbol as a neighbor, the recovered input symbol is removed as a neighbor from each of these encoding symbols and the degree of each such encoding symbol is decreased by one to reflect this removal.


codeword
input
1
1
1
0
1
?
0
1
1
1
1
?


input
codeword
1
1
1
0
1
1 = 0 XOR 1
1
1
1
1


input
codeword
1
1
1
0
1
1
1
1
1
1


input
codeword
1
1
1
0
1
0
0
1
0 = 1 XOR 1
1

2.6 Degree Distribution

For all d, ρ(d) is the probability that an encoding symbol has degree d. Each encoding symbol has a degree chosen independently from a degree distribution. As we now develop, the random behavior of the LT process is completely determined by the degree distribution $\rho(\ )$, the number of encoding symbols *k,* and the number of input symbols *n.* Our objective is to design degree distributions that meet the following two design goals. LT codes do not have a fixed rate and hence the desired property is that the probability of successful recovery is as high as possible while the number of encoding symbols required is kept small.

- As few encoding symbols as possible on average are required to ensure success of the LT process or in other terms the release rate of encoding symbols is low in order to keep the size of the ripple small and prevent waste of encoding symbols. Recall that the number of encoding symbols that ensure success of the LT process corresponds to the number of encoding symbols that ensure complete recovery of the data.

- The average degree is the number of symbol operations on average it takes to generate an encoding symbol. The average degree times K is the number of symbol operations on average it takes to recover the entire data or that the release rate of encoding symbols is high enough to keep the ripple from dying out.

(All-At-Once distribution): $\rho(1) = 1$.

This Distribution generates encoding symbols that have one neighbour each. Any received encoding symbol can immediately recover the associated information symbol. Once an encoding symbol is erased, however, the associated information symbol cannot be recovered. To prevent the failure, the transmitter needs to send more encoding symbols than 'k' , this distribution leads to waste of encoding symbols because of high release rate of encoding symbols.

In terms of LT codes, this corresponds to generating each encoding symbol by selecting a random input symbol and copying its value to the encoding symbol. The analysis of the classical balls and bins process implies that $n.\ \ln(n/\delta)$ encoding symbols are needed to cover all n input symbols with probability at least $1 - \delta$ with respect to the All-At-Once distribution.

This result can be easily modified to show that for any distribution the sum of the degrees of the encoding symbols must be at least $n.\ln(n/\delta)$ to cover all input symbols at least once. Thus, although the total number of symbol operations with respect to the All-At-Once distribution is minimal, the number of encoding symbols required to cover the n input symbols is an unacceptable $\ln(n/\delta)$ times larger than the minimum possible. Below we develop the Soliton distribution that ensures that just over 'k' encoding symbols with the sum of their degrees being $O(n.\ln(n/\delta))$ suffice to cover all n input symbols. Thus, both the number of encoding symbols and the total number of symbol operations are near minimal with respect to the Soliton distribution.

2.7 <u>Ideal Soliton Distribution</u>

The basic property required of a good degree distribution is that input symbols are added to the ripple at the same rate as they are processed. This property is the inspiration for the name Soliton distribution, as a Soliton wave is one where dispersion balances refraction perfectly.

The Ideal Soliton distribution is given by:

- p(i)= 1/k ,(for i=1)
- p(i) = 1/i(i-1) for all i = 2, . . . , k.

  Figure 7 shows the Ideal Soliton distribution with various k values.

Drawback:

- Expected Ripple size=1

Smallest variance can cause it to vanish .

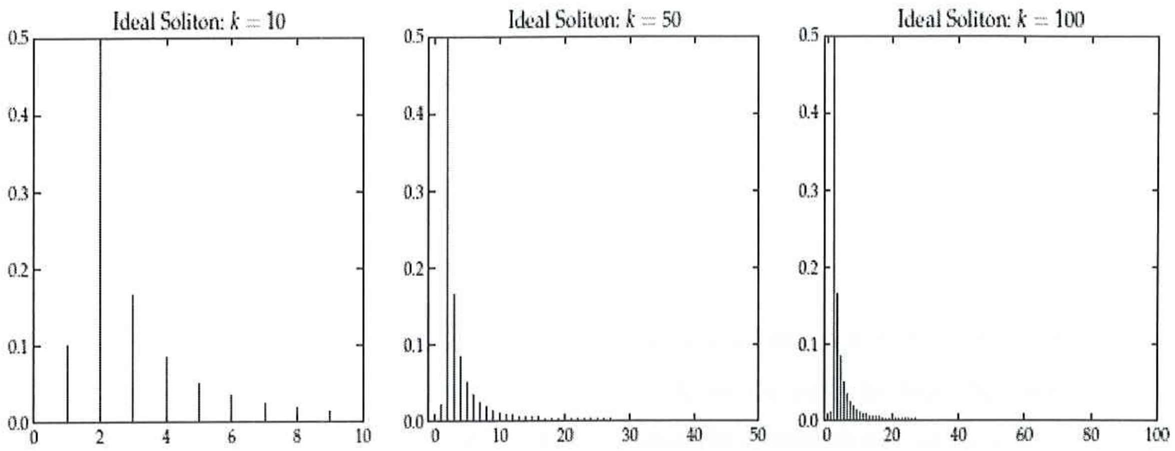


Figure 7 : Ideal Soliton Distribution for k =10,50,100

A desired effect is that as few released encoding symbols as possible cover an input symbol that is already in the ripple. This is because released encoding symbols that cover input symbols already in the ripple are redundant. Since released encoding symbols cover a random input symbol from among the unprocessed input symbols, this implies that the ripple size should be kept small at all times. If the ripple vanishes before all n input symbols are covered then the overall process ends in failure. The ripple size should be kept large enough to ensure that the ripple does not disappear prematurely. The desired behavior is that the ripple size never gets too small or too large.

(Uniform release probability):

For the Ideal Soliton distribution, $r(L) = 1/k$ *for all* $L = k, \ldots, 1$.

Proof: For $L = k$ all encoding symbols of degree one are released, and an encoding symbol is of degree one with probability $1/k$, and thus the statement is true for $L = k$.

Suppose the expected behavior of the LT process is its actual behavior. Then, the Ideal Soliton distribution works perfectly, i.e. exactly k encoding symbols are sufficient to cover each of the k input symbols exactly once. This is because if there are k encoding symbols, then exactly one encoding symbol is expected to be released initially and exactly one encoding symbol is expected to be released each time an input symbol is processed. If the expected behavior actually happens, then there is always exactly one input symbol in the ripple, and when this symbol is processed the released encoding symbol covers an input symbol among the unprocessed input symbols to reestablish this condition, etc.

However, this heuristic analysis makes the completely unrealistic assumption that the *expected behavior is the actual behavior*, and this is far from the truth. In fact, the Ideal Soliton distribution works very poorly in practice because the expected size of the ripple is one, and even the smallest variance causes the ripple to vanish and thus the overall process fails to cover and process all input symbols.

Recall that for the All-At-Once distribution it takes on average $k \,.\, \ln(k)$ encoding symbols to cover all input symbols with constant probability. It is interesting that the number of symbol operations for the Ideal Soliton distribution and for the All-At-Once distribution coincide, although the number of encoding symbols is quite different. The intuition for this is that for any distribution the sum of the degrees of all the encoding symbols needs to be around $k \,.\ln(k)$ in order to cover all the input symbols, but the Ideal Soliton distribution compresses this into the fewest number of encoding symbols possible. Thus, the total amount of computation is the same in both cases (since it is proportional to the sum of the degrees of all the encoding symbols).
However, the total amount of information that needs to be received (which is proportional to the number of encoding symbols) is compressed to the minimum possible with the Ideal Soliton distribution.

In practice, however, the Ideal Soliton distribution shows poor performance. Even a small variation can make the ripple exhausted in the middle of decoding, which leads to failure. Therefore, we need a distribution that ensures the ripple of large expected size enough to enable stable decoding as well as has the nice property of the Ideal Soliton distribution that maintains the expected ripple size constant in order not to waste encoding symbols.

2.8 <u>Robust Soliton Distribution</u>

Till now, we have studied and concluded why the Ideal Soliton Distribution is not suitable for practical implementation. This topic will help us understand how a modified distribution (Robust Soliton) can be used to implement the degree distribution which is further used for the rest of the analysis.

The Robust Soliton Distribution ensures that the expected size of the ripple is large enough at each point in the process so that it never disappears completely with high probability. On the other hand, in order to minimize the overall number of encoding symbols used, it is important to minimize the expected ripple size so that not too many released encoding symbols redundantly cover input symbols already in the ripple.

't' is added to ideal distribution 'p' to increase the expected ripple size.

$R=c.\ln(k/\delta)$ (Expected ripple size)

- $t(i) = R/ik$ for $i = 1, \dots, k/R - 1$
- $t(i)=R\ln(R/\delta)/k$ for $i = k/R$
- $t(i)=0$ for $i = k/R + 1, \dots, k$

The Robust Soliton Distribution is given by:

$\mu(i) = (p(i) + t(i))/\beta.$

Where $\beta = \sum_{i=1}^{k}[\, p(i) + t(i)]$

Figure 8 shows Robust Soliton Distribution for various values.

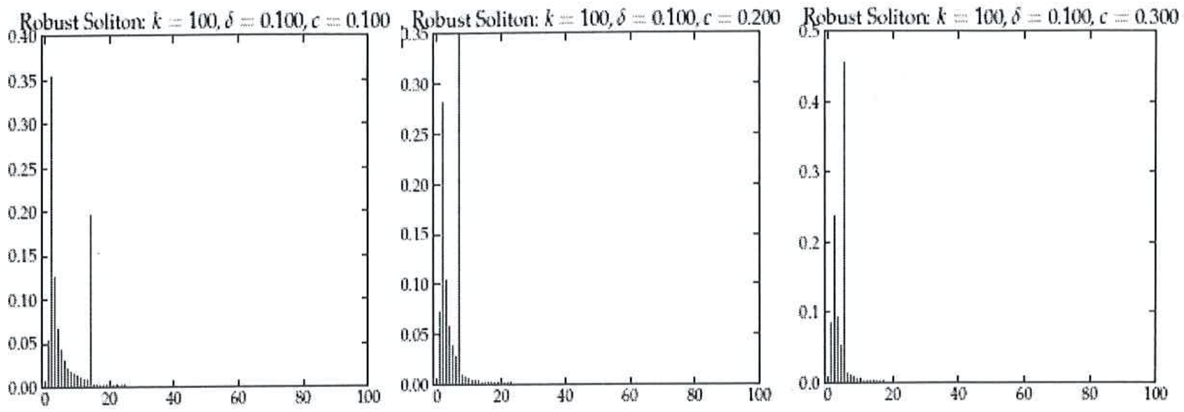


Figure 8 : Robust Soliton Distribution for k=100 and c=0.1,0.2,0.3

Let δ be the allowable failure probability of the decoder to recover the data for a given number 'k' of encoding symbols. The idea here is to design the distribution so that the expected ripple size is about $\ln(k/\delta)\sqrt{k}$ throughout the process.

The intuition is that the probability a random walk of length 'k' deviates from its mean by more than $\ln(k/\delta)\sqrt{k}$ is at most δ. As we describe below, it turns out this can be achieved using $K = k + O(\ln2(k/\delta)\sqrt{k})$ encoding symbols.

The idea in the Robust Soliton distribution is that a distribution T that increases the expected ripple size is added to the Ideal Soliton distribution Q so that the resulting degree distribution has larger expected ripple size than P while still maintaining approximately uniform release probability.

The intuition for the supplement $t(\,)$ is the following:

At the beginning, $t(1)$ ensures that the ripple starts off at a reasonable size. Consider the process in the middle. Suppose that an input symbol is processed and L input symbols remain unprocessed. Since the ripple size decreases by one each time an input symbol is processed, on average the ripple should be increased by one to make up for this decrease. If the ripple size is R then the chance that a released encoding symbol adds to the ripple is only $(L - R)/L$. This implies that at this point it requires $L/(L-R)$ released encoding symbols on average to add one to the ripple. It is possible to verify that the release rate of encoding symbols of degree i for i within a constant factor of k/L make up a constant portion of the release rate when L input symbols remain unprocessed. Thus, if the ripple size is to be maintained at approximately R, then the density of encoding symbols with degree $i = k/L$ should be proportional to

$$\frac{L}{i(i-1)\cdot(L-R)} = \frac{k}{i(i-1)\cdot(k-iR)}$$

$$= \frac{1}{i(i-1)} + \frac{R}{(i-1)(k-iR)} \approx \rho(i)+\tau(i),$$

for $i = 2, \ldots, k/R-1$.

The final spike $t(k/R)$ ensures that all the input symbols unprocessed when $L = R$ are all covered. This is similar to simultaneously releasing $R\ln(R/\delta)$ encoding symbols when R input symbols remain unprocessed to cover them all at once. Thus, the wastage caused by releasing enough encoding symbols to cover each of these input symbols at least once is only a small fraction of the total number 'n' of input symbols.

2.9 Channel Noise and Fading

Till now we have discussed in detail about the LT codes and how their practical implementations can be realized. However the channel introduces certain characteristics like noise which introduces errors in the received data. Thus, in the following section we shall study about the problems that are encountered during real-time transmission.

2.9.1 Additive White Gaussian Noise

Additive white Gaussian noise (AWGN) is a channel model in which the only impairment to communication is a linear addition of wideband or white noise with a constant spectral density (expressed as watts per hertz of bandwidth) and a Gaussian distribution of amplitude. The model does not account for fading, frequency selectivity, interference, nonlinearity or dispersion.

The AWGN Channel block adds white Gaussian noise to a real or complex input signal. When the input signal is real, this block adds real Gaussian noise and produces a real output signal. When the input signal is complex, this block adds complex Gaussian noise and produces a complex output signal. This block inherits its sample time from the input signal.

The AWGN channel is a good model for many satellite and deep space communication links. It is not a good model for most terrestrial links because of multipath, terrain blocking, interference, etc. However, for terrestrial path modeling, AWGN is commonly used to simulate background noise of the channel under study, in addition to multipath, terrain blocking, interference, ground clutter and self interference that modern radio systems encounter in terrestrial operation.

In matlab AWGN can be implemented using the AWGN function.

Y = AWGN(X,SNR,SIGPOWER) when SIGPOWER is 'measured', AWGN measures the signal power before adding noise.Here X is the signal to be transmitted.

2.9.2 Rayleigh Fading

Rayleigh fading is a statistical model for the effect of a propagation environment on a radio signal, such as that used by wireless devices. Rayleigh fading models assume that the magnitude of a signal that has passed through such a transmission medium (also called a communications channel) will vary randomly, or fade, according to a Rayleigh distribution — the radial component of the sum of two uncorrelated Gaussian random variables.

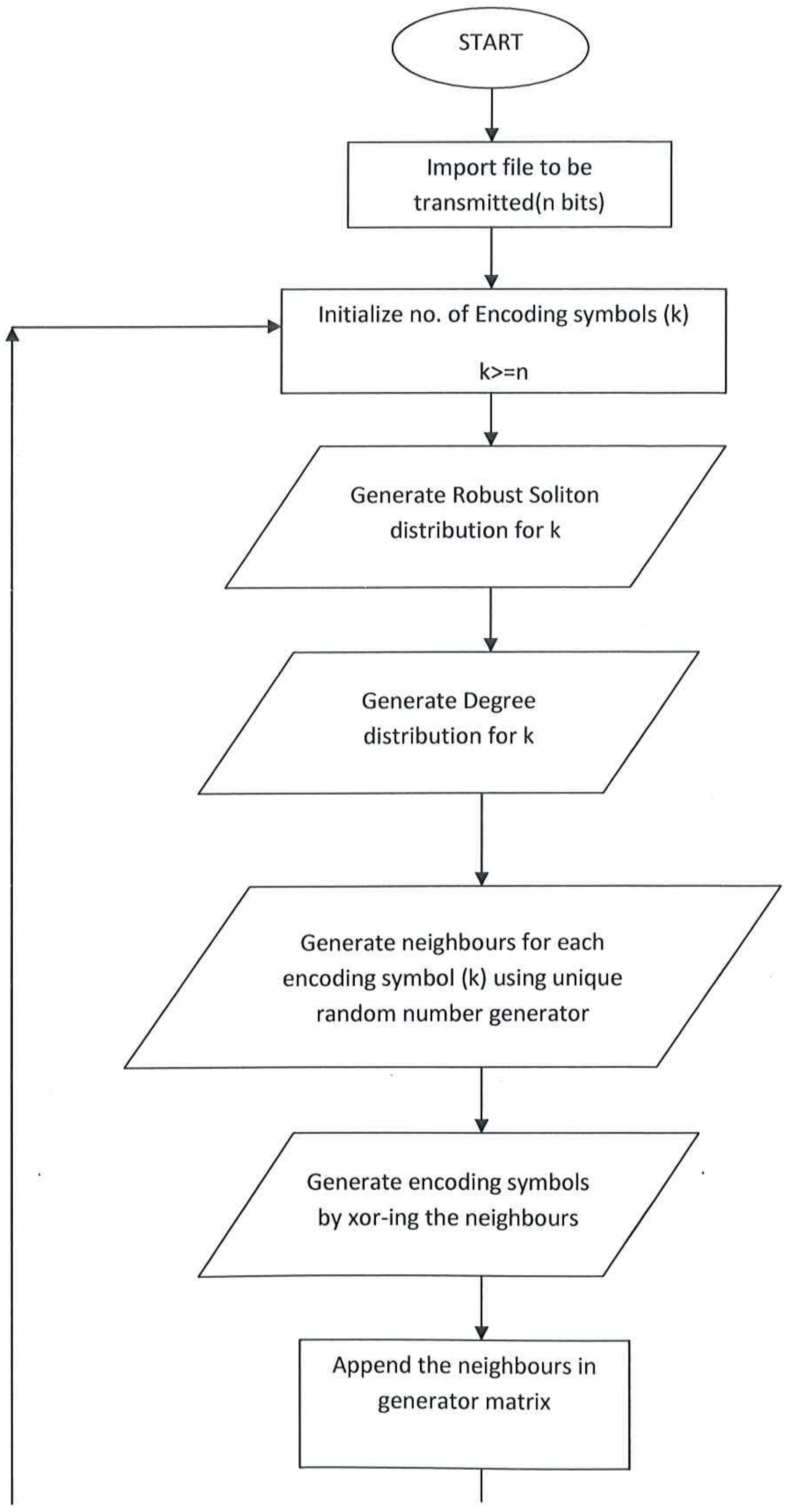
Rayleigh fading is viewed as a reasonable model for tropospheric and ionospheric signal propagation as well as the effect of heavily built-up urban environments on radio signals. Rayleigh fading is most applicable when there is *no dominant propagation* along a **line of sight(LOS)** between the transmitter and receiver. If there is a dominant line of sight, Rician fading may be more applicable.

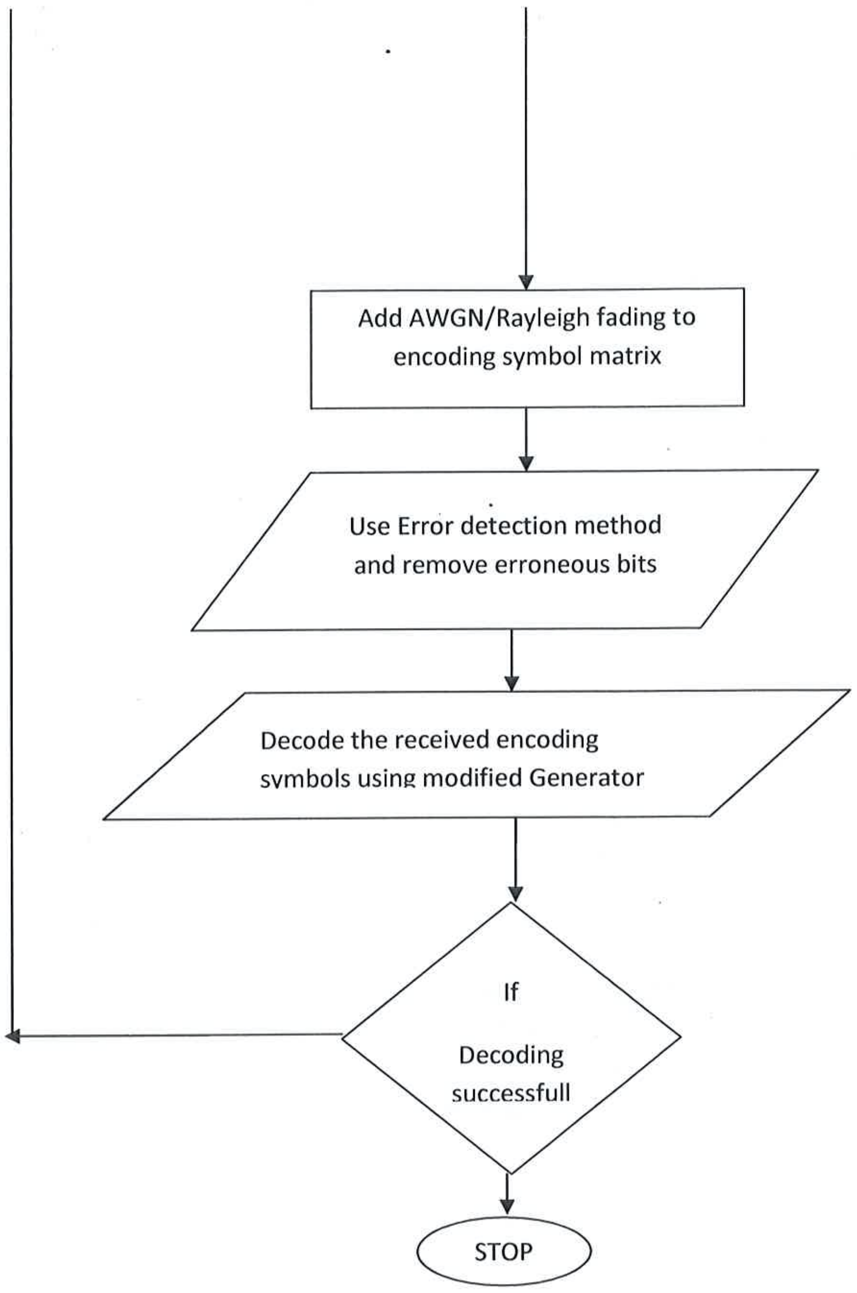
Thus, in case of multipath propagation, that is when data is being concurrently sent to many receivers, the signal experiences fading such that by the time it reaches the receiver, errors are introduced in the data because of the channel's inherent properties thereby leading to erroneous data being received at the receiver end. Rayleigh fading thereby becomes of paramount importance in case of multipath propagations. Thus, at several instances the transmitter is required to resend the data to several receivers that have lost or received corrupted data. In the real time data transmission that we have implemented, we have tried to realize the concept of a fading channel and make the required modifications in order to ensure the successful reception of the data.

In matlab ,rayleigh fading is implemented using the function RAYLEIGHCHAN.

CHAN = RAYLEIGHCHAN(TS, FD) constructs a frequency-flat ("single path") Rayleigh fading channel object. TS is the sample time of the input signal, in seconds. FD is the maximum Doppler shift, in Hertz. We can model the effect of the channel CHAN on a signal X by using the syntax :Y = FILTER(CHAN, X).

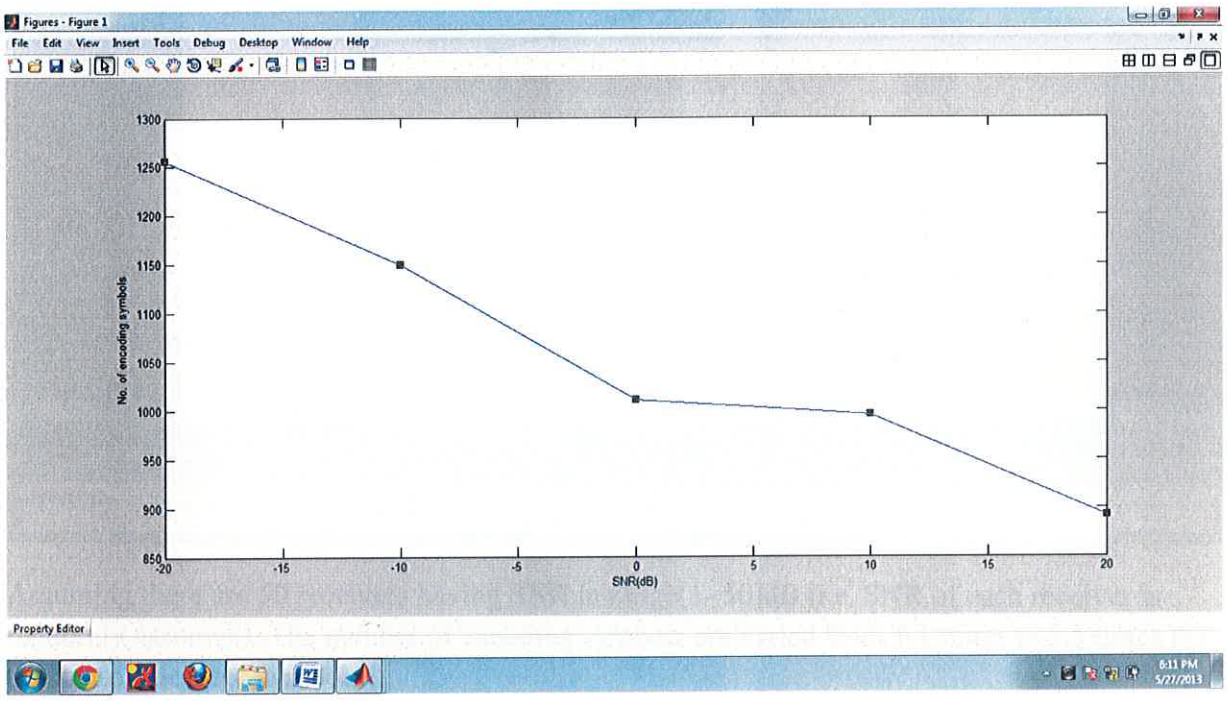
CHAPTER III: Methodology


START
Import file to be transmitted(n bits)
Initialize no. of Encoding symbols (k)
k>=n
Generate Robust Soliton distribution for k
Generate Degree distribution for k
Generate neighbours for each encoding symbol (k) using unique random number generator
Generate encoding symbols by xor-ing the neighbours
Append the neighbours in generator matrix


Add AWGN/Rayleigh fading to encoding symbol matrix
Use Error detection method and remove erroneous bits
Decode the received encoding symbols using modified Generator
If Decoding successfull
STOP

CHAPTER IV: Results

Graph depicting variation of no. of Encoding symbols w.r.t SNR (AWGN & Rayleigh Fading added)

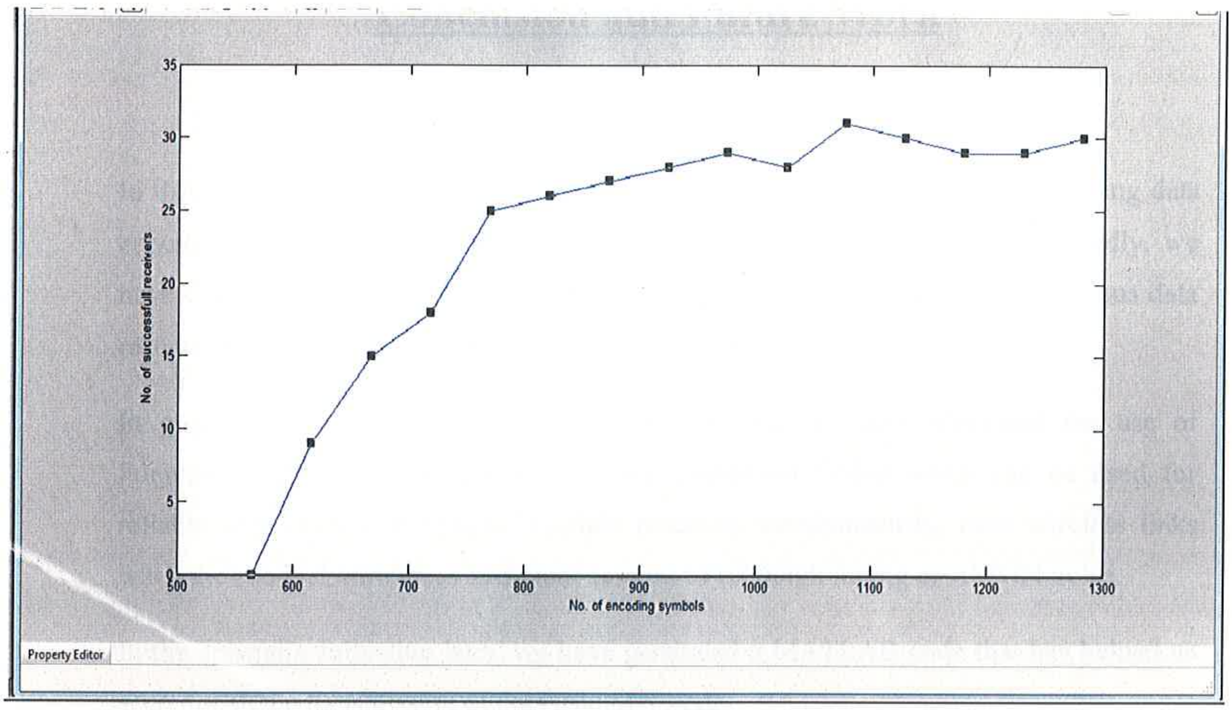


Through this curve obtained on Matlab, we conclude that Fountain codes requirement of number of symbols to transmit a given data stream, varies inversely as SNR for a given medium. As network characteristics improve there is minimal requirement of overheads(extra number of encoding symbols).

If similar data stream was to be transmitted through standard TCP algorithms , each and every transmitted bit would have required a corresponding acknowledgement and retransmission in several cases. Such an approach would have caused network flooding with large number of data packets and hence malfunctioning.

This approach hence reduces network load significantly for large stream data delivery and provides efficient technique for data transmission.

Graph depicting variation of number of successful receivers w.r.t. number of encoding symbols (SNR assumed randomly)



Assuming there are 50 receivers having SNR in range[1- 50]dB (i.e. SNR of each receiver is randomly assumed). The number of encoding symbols are varied from 1.1 times to 2.5 times the number of message symbols.

Conclusion and Future Work

In this work, we have firstly studied the drawbacks encountered when transmitting data concurrently to many receivers by using traditional protocols like TCP. Secondly, we have examined the channel induced characteristics like noise that lead to erroneous data reception at the receiver end when data is being transmitted over wireless links.

In order to counter the above mentioned problems, we have illustrated the use of Fountain Codes, or in particular the Luby Transform Codes which can be used for reliable transmission of data to multiple receivers simultaneously, over wireless links when the channel introduces sufficient amount of Rayleigh fading or AWGN noise.

In the appendix following later, we have generated a MATLAB code that has helped us accomplish the transmission of data using LT code.

In relation to the future scope of this work, we can extend upon this by using Raptor codes, an enhancement of the Luby Transform codes. Raptor codes are another class of Fountain codes as described in the preceding chapters that are more suitable for applications like streaming data , voice over IP etc. Moreover, the speed with which the data is transmitted using this coding scheme is considered good for such applications.

Bibliography

- Michael Luby,
  LT Codes, Digital Fountain, Inc. luby@digitalfountain.com

- Gauri Joshi, Joong Bum Rhim, John Sun, Da Wang
  Fountain Codes,6.972 PRINCIPLES OF DIGITAL COMMUNICATION II,December 7, 2010

- Amin Shokrollahi, *Senior Member, IEEE*
  Raptor Codes, IEEE TRANSACTIONS ON INFORMATION THEORY, VOL. 52, NO. 6, JUNE 2006

- D.J.C. MacKay,
  Capacity Approaching Codes, Design and implementation special section

Appendices

SOURCE CODE:

```
clc;
clear all;

                                    % Importing a file and defining
                                            % constants



img=randint(1,513,[0,1]);
d=0.01;
c=.1;
h=1;

for snr=-20:10:20



        var=zeros(1,50);                            %max no. of values for v
         cntr=1;
         summ=0;
         coun=0;

         for v=.8:0.1 :2.5
       summ=summ+v;
       coun=coun+1;
       clear e*;
       clear g*;
       clear col*;
       k=floor(v*length(img));
           a=log(k/d)*(k^0.5);
         r=c*a;

                                      %Generating Robust Soliton


         for  i=1:k
             if( i>=1 && i<k/(r-1) )
                 t(i)=r/(i*k);

             else if ( i==k/r )
                     t(i)=r*log(r/d)/k;
        else
            t(i)=0;


        end;
    end;
end;






for j=1:k
    if(j==1)
        p(j)=1/k;
    else
        p(j)=1/(j*(j-1));
    end
end




b=0;
for j=1:k
    b=b+(p(j)+t(j));
end

for j=1:k
    u(j)=(p(j)+t(j))/b;
end

x=rand(1,k);

                                        % Generating Degree


for i=1:k
    s=0;
    for j=1:k
        if(x(i)>s && x(i)<(s+u(j)))
            deg(i)=j;
            if(deg(i)>length(img))
                deg(i)=length(img);
            end
        end
        s=s+u(j);
    end
end



                              %Generating unique Random number
```

```
a=deg;
gm=zeros(length(img),k);
for i=1:k
    x=randint(1,1,[1,length(img)-1]);
    y=randint(1,1,[0,length(img)-1]);

    gm(y+1,i)=1;
    ngh(1)=y;

    if( deg(i)>=2 )

        for j=2:deg(i)
            ngh(j)=mod((ngh(j-1)+x),length(img));
            gm(ngh(j)+1,i)=1;
        end
    end
end;


enc=zeros(1,k);

for i=1:k
    m=1;
    index=zeros(1,k);
    x=0;
    for j=1:length(img)


        if( gm(j,i)==1 )
            index(1,m)=j;
            m=m+1;

        end
    end
    f=0;

    for s=1:length(index)
        if( index(1,s)== 0)
            f=s-1;
            break;
        end
    end


                                % Generating encoding symbols
    for n=1:f

        x=xor ( x,img(index(1,n)) );
        enc(1,i)=x;


    end
```

```
end

                                        % AWGN added and transmitted

enct=zeros(1,length(enc));
for i=1:length(enc)
    if(enc(i)==0)
        enct(i)=-5;
    else
        enct(i)=5;
    end
end



encnoise=awgn(enct,snr,'measured');
                                                      % Adding rayleigh fading
  ts=length(img);
   fd=0;
  chan = rayleighchan(ts,fd);
  rchnl = filter(chan,encnoise);


      meanval = mean(rchnl);

encr=zeros(1,length(encnoise));

for j=1:length(encnoise)
    if(meanval < rchnl(j))
        encr(j)=1;
    else
        encr(j)=0;
    end
end


% Comparing encoding bits with and without noise for error detection

j=1;
for i=1:length(enc)
    if(enc(i)~=encr(i))
        errr(j)=i;
        j=j+1;
    end
end
lng=j-1;
```

```
                    %Removing errored bits from generator matrix

gmfinal=gm;
for i=1:lng
    for j=1:length(img)
        gmfinal(j,errr(i))=0;
    end
end


                               %  Decoding after noise addition


a=sum (gmfinal);
gm11=gmfinal;
dnc11=encr;


msg11=zeros(1,length(img) );
check=zeros(length(img),k);

tf=0;



for i=1:k

    while(~tf && a(i)==1)


        x=1;
        for l=1:k
            if(a(l)==1)
                column(x)=l;
                x=x+1;
            end
        end


        for i=1:x-1
            for j=1:length(img)
                if(gm11(j,column(i))==1)
                    row(i)=j;
                end
            end
        end


        for i=1:x-1
            msg11( row(i) )=dnc11( column(i) );

        end;
```

```
                for j=1:x-1
                    gm11(row(j),column(j))=0;
                end

                for j=1:x-1
                    b=1;

                    for i=1:k

                        if ( gm11(row(j),i)==1 )
                            clm(b)=i;
                            b=b+1;
                        end
                    end

                    for m=1:b-1
                  dnc11( clm(m) )=xor( dnc11(clm(m)),msg11( row(j) ) );
                    end


                    for l=1:b-1
                        gm11( row(j),clm(l) )=0;
                    end

                    a=sum(gm11,1);
                end

                tf=isequal(gm11,check);
                i=1;
            end
        end


     ans=isequal(msg11,img);
        if(ans==1)
            v;
        else

            var(cntr)=v;
            cntr=cntr+1;


        end

    end


    mb=0;
    for i=1:cntr-1
        mb=mb+var(i);
    end
    SNR(h)=snr;
    avg(h)=floor((summ-mb)/(coun-cntr+1)*length(img))
    h=h+1;

end
  plot(SNR,avg);
```