Tornado Codes and Luby Transform Codes

Ashish Khisti

October 22, 2003

1 Introduction

A natural solution for software companies that plan to efficiently disseminate new software over the Internet to millions of users simultaneously is to use multicast or broadcast transmission. Unicast protocols are based on receiver initiated retransmission requests and do not scale with the number of users. In this paper, we summarize the initial approach taken by Digital Fountain, a company founded by Michael Luby and Jay Goldin in 1998 to solve the multicast problem.

We first describe the charasteristics of an ideal multicast protocol as proposed by Digital Fountain[2]. We then describe Tornado Codes, followed by Luby Transform (LT) Codes. These codes were developed as a step towards approximating the ideal protocol. Tornado codes first appeared in a technical report [3] in 1997 and were also published in [4]. LT codes were proposed in 2002 [5]. Both these codes have now been superseded by Raptor Codes, which will be the topic of next week.

2 Ideal Digital Fountain Protocol

Consider an application, where a server wishes to distribute a file of k equal length packets to a very large number of clients. The main requirements of an ideal protocol for this application as outlined in [2] are:

1. **Reliable:** The file is guaranteed to be delivered in its entirety to all receivers.
2. **Efficient:** Both the total number of packets each client needs to receive and the amount of processing time on these packets should be minimal. Ideally the total time should be no more than what is required for a point to point communication link.

3. **On demand:** Clients may initiate the download at their discretion. Clients may sporadically be interrupted and continue the download at a later time.

4. **Tolerant:** The protocol should tolerate a heteregeneous population of receivers, especially a variety of end-to-end packet loss rates and data rates.

In the ideal solution proposed in [2], the server encodes the original file into a stream of distinct encoding packets generated on the fly. It transmits packets whenever atleast one client is listening. Each client accepts packets until it receives k distinct encoding packets and efficiently reconstructs the original file. The server is analogous to a fountain of water that continuously produces a stream of water drops and hence the name.

3 Traditional Erasure Codes

A communication link over the Internet is often modelled as an erasure channel. This channel was first introduced by P. Elias[1] in 1956. In a typical erasure channel, input symbols are either received correctly or get erased with a probability p. Elias showed that the capacity of a memoriless erasure channel is $\log_2(|X|)(1-p)$ *bits/symbol*, where $|X|$ is the input alphabet size. Furthermore, he showed that a random linear code can achieve this capacity. This implies that at most $O(n^2)$ time is required for encoding and decoding, where n, denotes the block length.

Maximum distance separable (MDS) codes are practical codes that achieve the capacity of the erasure channel. A *(n,k,d)* MDS code, has a property that any k coordinates constitue an information set[6]. A receiver that receives any k symbols from a total of n symbols in each codeword can reconstruct the original message, provided it knows the position of the k received symbols. Reed Solomon (RS) codes are the most well-known MDS codes. These can be decoded in time $O(k^2)$, using algebraic methods such as list decoding.

Despite their popularity, RS codes are not suitable for bulk data distribution over the internet. The quadratic decoding time is unacceptable when data rates are of the order of Mbps [2]. Furthermore, typical RS code implementations have small block lengths such as the NASA standard $(255, 233, 33)$ code over F_{256}. This requires a large file to be segmented into many small blocks before transmission. Finally, since RS codes are block codes, they need to be designed for a specific rate. This requires that we need to estimate the erasure probability of the channel before hand. This is clearly not possible when multiple clients over different quality of channels are being served simulataneously.

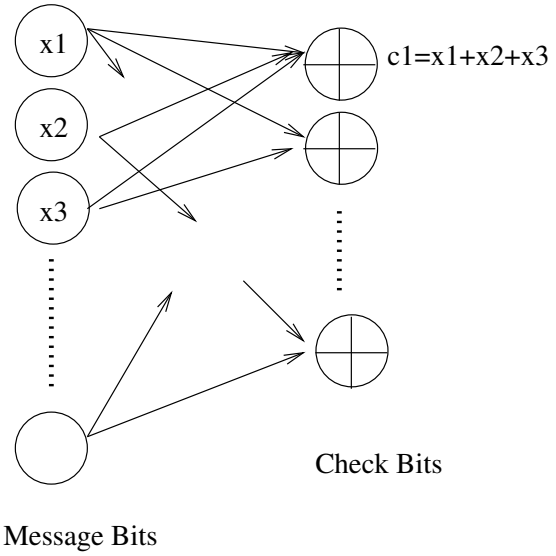

Figure 1: Irregular Bipartite Graph

4 Tornado Codes

Tornado codes are erasure block codes based on irregular spare graphs. Given an erasure channel with loss probability p, they can correct upto $p(1-\varepsilon)$ errors. They can be encoded and decoded in time proportional to $n\log(1/\varepsilon)$. Thus Tornado Codes has been primarily designed to speed up erasure codes over the internet. These codes can be designed over arbitrary alphabet size. In what follows we will consider a binary alpabet for simplicity.

4.1 Construction

Tornado codes are generated by cascading a sequence of irregular random bipartitie graphs. Unlike the case of LDPC codes, these graphs are equivalent to generator matrices. The operation of one such graph is shown in figure 1. The nodes on the left are known. The values of nodes on the right are computed by performing an XOR operation of the neighboring input nodes.

The overall code $C(B_0, B_1, ...B_m, \Lambda)$ is a cascade of bipartite graphs $B_0, B_1, ...B_m$ and Λ. See figure 2. The graph B_0 has n message bits as input and produces βn check bits. These form the input bits of B_1 and $\beta^2 n$ new check bits are formed. The graph B_i, has $\beta^i n$ input bits (from B_{i-1}) and produces $\beta^{i+1} n$ check bits. This sequence is truncated by a conventional rate $1-\beta$ erasure code Λ. The codeword

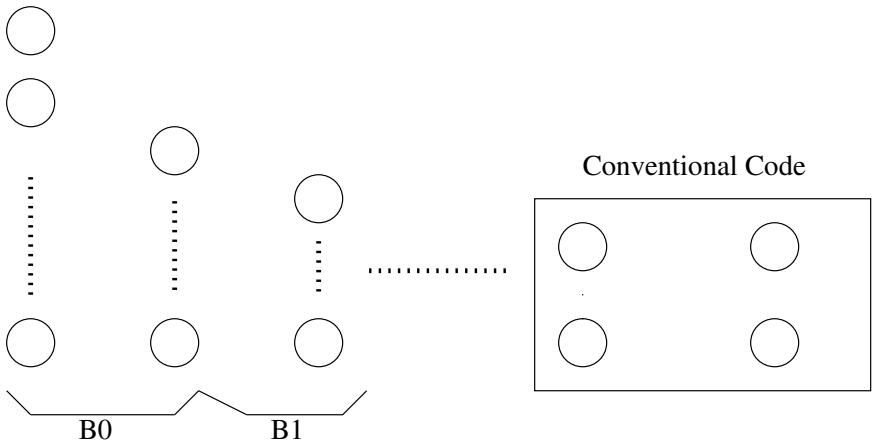

Figure 2: Cascaded Graph Sequence

consists of the n message bits and all the check bits produced at each stage of the cascade. It is thus a systematic code. The total number of check bits produced by this sequence is given by

$$\sum_{i=1}^{m+1} \beta^i n + \frac{\beta^{m+2} n}{1-\beta} = \frac{n\beta}{1-\beta}$$

The length of the codeword produced given n input bits is $n + \frac{n\beta}{1-\beta} = \frac{n}{1-\beta}$. The resulting code is a rate $1-\beta$ code for all values of m. The length of the cascade is chosen such that $\beta^{m+1} n \approx \sqrt{n}$. Note that Λ can be decoded and encoded in a time that is quadratic in the size of its input. We begin by using this decoding algorithm for Λ, to recover losses that occur within its bits. This will be successful if atmost β fraction of bits have been lost in Λ. If all the losses are recovered, we know the check bits of B_m. These could be used to recover any losses in the input bits of B_m. Since the input bits of B_i are the check bits of B_{i-1}, this recursion can be continued until all the input bits of B_0 are recovered. Furthermore, if we have a decoding algorithm that, knowing the check bits, can recover a loss of $\beta(1-\varepsilon)$ fraction in the input bits in each of B_i, in a time linear in the size of input,the overall code can recover from $\beta(1-\varepsilon)$ fraction of losses in a time that is linear in n.

4.2 Linear Time Decoding

The decoding algorithm is very simple. At each step, a right node is selected whose all but one neighbors are known. The missing neighbor is then computed by performing an XOR operation between the check bit and the known input bits. If at

any step, no such node is found then an error is declared. The algorithm terminates successfully when all the input bits are recovered.

Clearly, the algorithm is linear time, since at each step we recover one lost input bit. The algorithm is not optimal, since the algorithm can fail even when it would have been possible to recover the input bits through say gaussian elimination. The main contribution of ths paper is to design and analyze bipartite graphs for which this simplistic decoding algorithm recovers all the missing input bits from a loss fraction arbitrarily close to the capacity.

4.3 Analysis based on Degree Sequences

A random bipartite graph with n left nodes and $n\beta$ right nodes is specified by the following two sequences:

1. **Fraction of Edges of Degree i on left** $(\lambda_1, \lambda_2 ... \lambda_m)$: λ_i is the fraction of edges which are incident on a node of degree i one the left side of the graph.

2. **Fraction of Edges of Degree i on right** $(\rho_1, \rho_2 ... \rho_m)$: ρ_i is the fraction of edges which are incident on a node of degree i on the right side of the graph.

Let us define the following two polynomials over $x \in (0,1]$

$$\begin{aligned} \lambda(x) &= \sum_i \lambda_i x^{i-1} \\ \rho(x) &= \sum_i \rho_i x^{i-1} \end{aligned}$$

The main results of the analysis of the decoding processes are stated in the following three lemmas. In all these results,δ is the probability that each input bit is lost independently of all other input bits.

1. A necessary condition on δ, if the above simplistic decoding algorithm terminates successfully is that $\rho(1-\delta\lambda(x)) > 1-x$, for all $x \in (0,1]$

2. If $\rho(1-\delta\lambda(x)) > 1-x$, for all $x \in (0,1]$, then for all $\eta > 0$, the decoding algorithm terminates with atmost ηn message bits erased with probability $1 - n^{2/3} exp(-\sqrt[3]{n}/2)$.

3. If $\lambda_1 = \lambda_2 = 0$, then with probability $1 - O(n^{-3/2})$, the recovery processes restricted to the subgraph induced by any η fraction of the left nodes terminates successfully.

To derive the above results, the authors study the expected value behavior of the decoding process through a set of differential equations. They use tools of statistical mechanics to show that the variance of the process is small. The condition that $\lambda_1 = \lambda_2 = 0$, in lemma 3 is required since the authors use standard proofs on expander graphs to show that the expansion condition holds with high probability and the decoding algorithm terminates successfully.

4.4 Capacity Achieving Codes

Solving for the optimal degree sequences that maximizes the allowed loss probability δ is a non-trivial problem. Fortunately, for the erasure channel this problem has been solved and the optimal degree distributions are known.

We fix a positive integer D. Let $H(D) = \sum_{i=1}^{D} 1/i$, be the truncated harmonic sum. Thus for large D, $H(D) \approx \log(D)$. Let $\lambda_1 = 0$, and for $i = 2,3...D+1$, $\lambda_i = \frac{1}{H(D)(i-1)}$. Also for all $i \geq 1$, let $\rho_i = \frac{e^{-\alpha}\alpha^{i-1}}{(i-1)!}$. Here the choice of alpha is not arbitrary but depends on the choice of D. Let there be E edges on the graph, k input bits on the left and $k\beta$ check bits on the right. Then it follows from definition that

$$\begin{aligned} k = \sum_i \frac{\lambda_i E}{i} &= \frac{EH(D)(D+1)}{D} \\ k\beta = \sum_i \frac{\rho_i E}{i} &= \frac{E\alpha e^{\alpha}}{e^{\alpha}-1} \end{aligned}$$

It follows that $\frac{\alpha e^{\alpha}}{e^{\alpha}-1} = \beta\frac{H(D)(D+1)}{D}$. Thus α, depends on D and cannot be independently chosen. Also note that we allow $\rho_i > 0$ for all i. In practice, since the number of edges E is finite, we will have to truncate the expression for a large i. This will not make a difference if E is sufficiently large. Also, note that $\rho(x) = e^{\alpha(x-1)}$ and $\lambda(x)$ is the expansion of $-\log(1-x)$ truncated to the D^{th} term and scaled so that $\lambda(1) = 1$.

Lemma With the above choice of $\rho(x)$ and $\lambda(x)$, we have $\rho(1-\delta\lambda(x)) > 1-x$, for all $x \in (0,1]$if $\delta \leq \frac{\beta}{1+1/D}$.

The proof follows by direct substitution in the expressions of $\rho(x)$ and $\lambda(x)$ derived earlier. There is a slight technical difficulty in proving that this value of δ is sufficient to guarantee success in decoding. The reason is that result (3) in 4.3, requires that $\lambda_2 = 0$, whereas our construction has $\lambda_2 \neq 0$. To alleviate this difficulty, we isolate a small fraction of check nodes to form a set S and generate a graph B_2 in which each input bit node had three edges incident on randomly chosen elements of S. The remaining check nodes in S' and the input nodes connect using the degree sequences stated before. By result (2) in 4.3, we can argue that the size

of S can be made vanishingly small and successful decoding is guaranteed. Note that by choosing $D = 1/\varepsilon$, we see that $(1-R)(1-\varepsilon)$, fraction of erasures can be corrected in a time proportional to $n\log(1/\varepsilon)$.This proves the main result of the paper.

4.5 Computing Degree Sequences using Linear Programming

Although the optimal degree sequences are known for the erasure channel, the authors provide an ad-hoc linear programming method to compute good degree sequences. This kind of approach can be used to find degree sequences for other channels after suitable conditions have been found.

The linear programming problem is posed as follows: Fix $\lambda(x)$ and δ. The objective is to find $\rho_m \in M$, where M is a fixed set of positive integers.

Let $x_i = \frac{i}{N}$ for $i = 1, 2..N$ for some large N. Minimize $\sum_i(\rho(1-\delta\lambda(x_i))+x_i-1)$ subject to $\rho_i > 0$ and $\rho(1-\delta(x_i)) > 1-x_i$. This solution for $\rho(x)$ is feasible if $\rho(1-\delta\lambda(x)) > 1-x$ for all $x \in (0,1]$. Once a feasible $\rho(x)$ is found, the largest δ is found through a binary search.

The authors also propose an extension to this method that uses the dual condition $\delta\lambda(1-\rho(y)) < 1-y$. Once the optimal value of $\rho(x)$ is found for a fixed λ, it can be used to improve our choice of λ using this dual condition.

4.6 Practical Considerations

Tornado Codes implemented in practice have much fewer cascade graphs than that suggested by theoretical analysis. For example Tornado Z codes only have three cascaded graphs. These are rate 1/2 codes that map an input message of 640K packets into 1280K packets. Each packet is 256 bytes. The first layer has 640K nodes on the left and 320K nodes on the right. The second and third layers have 160K nodes on the right. The first and second graphs have the heavy-tail/poisson distribution as noted suggested in section 4.4. The third graph cannot use a conventional quadratic time code since it has many more than $\sqrt{n}$ nodes. The authors claim than linear programming has been used to find the degree seqence for this graph and a "double heavy tail" where $\lambda'(x) = \lambda(x^2)$ distribution is used

The assumption of independent erasures on each symbol is crucial in the analysis of Tornado Codes. If this were not to hold, then one cannot argue that there would be a δ fraction of loss in each graph when there is a δ fraction of loss overall. In practice the internet is a bursty channel and this assumption does not hold. So it is natural that the practical implementations only require a small number of cascades. Also the size of Tornado codes is very large and this introduces large latencies in encoding and decoding. The time is linear in block length as opposed to

the dimension and this can be disadvantageous for low rate codes. Like RS codes these are also block codes cannot optimally serve a heterogenous quality of users. Finally when the rate and n is fixed, the number of encoding packets that can be generated is also fixed. In other words, while tornado codes, do improve the encoding and decoding efficiency over RS codes, they are not good approximations to the Digital Fountain protocol.

5 LT Codes

Luby Transform (LT) codes were proposed by Michael Luby in 2002[5] as a better approximation to the digital fountain approach. Unlike Tornado Codes, these codes are rateless. Their design does not depend on the estimate of erasure probability of the channel, so they can simultaneously serve a heterogenous population of receivers efficiently. Furthermore distinct encoding symbols can be generated on the fly by the server as needed.

Suppose the original file has n message symbols. The receivers can reconstruct these message symbols with probability $1-\delta$ when any $n+O(\sqrt{n}\log^2(n/\delta))$ symbols have been received. The time for encoding each symbol is proportional to $O(\log(n/\delta))$. The time for decoding each symbol is proportional to $O(n\log(n/\delta))$. Thus LT codes have higher complexity than tornado codes. Just as in the case of Tornado codes, we will consider LT codes over GF(2). Extensions to higher alphabets is straightforward.

5.1 Encoding LT Codes

Each encoding symbol is generated independently of all other symbols by the following process:

1. Generate a random number d from the degree distribution $\rho(d)$
2. Randomly select a message node incident on each of the d edges. The value of the encoding symbol is the XOR of the neighboring encoding bits.

Since the encoding symbols are generated on the fly, how does the decoder know which message nodes are the neighbors of a particular encoding symbol? Luby suggests that one solution is to explicitly include this information as an additional overhead in the packet. Another possibility is to replicate the pseudorandom process at the receiver by supplying it with the suitable seed and/or keys.

5.2 Decoding LT Codes

The decoding process is virtually same as that of Tornado Codes. However we introduce some terminology that will be used in the analysis later. When the decoding process initiates all message symbols are *uncovered*. At the first step, all degree one encoding symbols get *released* to cover their unique neighbor. These set of covered message symbols that have not been processed yet form a *ripple*. At each subsequent set one message symbol from the ripple is selected randomly and *processed*. It is removed as a neighbor of all encoding symbols. Any encoding symbol that now has degree one is now released and its neighbor is covered. If the neighbor is not in the ripple it gets added to the ripple. The process ends when the ripple is empty. It fails if atleast one message symbol is uncovered.

5.3 Analysis of $\rho(1) = 1$

In this section, we analyze the special case in which all encoding symbols has degree 1. Suppose $N = k\log(k/\delta)$ encoding symbols are used. Note that the probability that any message symbol is not covered is given by $(1-1/k)^{k\log(k/\delta)} \approx \delta/k$. Thus by union bound estimate the probability that every message symbol is covered is greater than $1-\delta$. Thus $k\log(k/\delta)$ degree one encoding symbols are sufficient to guarantee that all message symbols are covered with high probability. This number is unacceptably large. This analysis also shows that the total number of randomly incident edges must be atleast $k\log(k/\delta)$ to cover all message symbols. This fact will be used in the next section.

5.4 Analysis of Soliton distribution

In this section, we consider the distribution: $\rho(i) = \frac{1}{i(i-1)}$ for $i = 2,3..k$ and $\rho(1) = 1/k$. We first define the following distributions:

1. q(i,L) is the probability that an encoding symbol of degree i is released when L symbols remain unprocessed. It is clear that

$$q(i,L) = \begin{cases} 1 & \text{i =1, L = k} \\ \frac{\binom{i}{i-2}\binom{2}{1}\binom{L}{1}\binom{k-L-1}{i-2}(i-2)!}{\binom{k}{i}i!} & \text{i = 2..k, L = k-i+1..1} \\ 0 & \text{otherwise.} \end{cases}$$

   Note that for the particular degree i encoding symbol to be released, we need $i-2$ edges to be deleted when $k-(L+1)$ message symbols are processed, the second last one is deleted when $(k-L)^{th}$ symbol is processed and the last edge to be a neighbor of one of the L unprocessed symbols.

2. r(i,L) is the probability that a node of degree i is chosen *and* is released when L message symbols remain unprocessed.

$$r(i,L) = \rho(i)q(i,L) = \frac{\binom{L}{1}\frac{(k-1-L)!}{(k-L-1-(i-2))!}}{\frac{k(k-1)!}{(k-i)!}}$$

3. r(L) is the probability that an encoding symbol is released when L message symbols are unprocessed. Clearly $r(L) = \sum_i r(i,L)$. To calculate this sum, note that $kr(i,L)$ is the probability of the event that the $(i-1)^{th}$ ball lands in one of the L designated bins (other others being empty), when $i-1$ balls are being randomly thrown into $k-1$ bins and each bin is removed as soon as a ball lands on it. The outcomes are mutually exclusive for each i and hence $\sum_i kr(i,L) = 1$. Thus we have $r(L) = 1/k$.

At each step in the process we expect one encoding symbol to be released. Thus only k encoding symbols are expected to be sufficient for the k message symbols. Note that the soliton distribution implies that the average degree of each encoding symbol is $\log(k)$. Thus the expected number of edges is $k\log(k)$. The soliton distribution minimizes the number of average number of encoding symbols while keeping the average number of nodes the same. Unfortunately, this distribution does not work well in practice, since the average size of the ripple is one and is very sensitive to variations. So there is a need to create a robust soliton distribution.

5.5 Robust Soliton Distribution

The robust soliton distribution has two extra parameters c and δ. It is designed to ensure that the expected size of the ripple is $R = c\log(k/\delta)\sqrt{k}$. Define

$$\tau(i) = \begin{cases} R/ik & \text{if i = 1,2..k/R-1 ,} \\ R\log(R/\delta)/k & \text{if i = k/R} \\ 0 & otherwise. \end{cases}$$

The robust soliton distribution is obtained by normalizing $\rho(\cdot)$ and $\tau(\cdot)$.

$$\begin{aligned} \psi &= \sum_{i=1}^{k} \rho(i) + \tau(i) \\ \mu(i) &= (\rho(i) + \tau(i))/\psi \text{ for i = 1,2..k} \end{aligned}$$

The intuition behind this choice of $\tau(\cdot)$ is as follows. Initially we need to generate a ripple of size R. Hence $\tau(1) = R/k$ is chosen, so that about R encoding

symbols will have degree one. As the decoding process continues,consider the step when L message symbols are unprocessed. It is possible to argue that the most likely degree of encoding symbols that get released at this stage is k/L. Since the ripple is of size R, the probability that any release adds a message symbol to the ripple is $(L-R)/L$. So we need on an average $L/(L-R)$ releases to add one symbol to the ripple. Thus the fraction of degree $i = k/L$ symbols should be proportional to

$$\begin{aligned} \frac{L}{(L-R)i(i-1)} &= \frac{k}{i(i-1)(k-iR)} \\ \frac{1}{i(i-1)} + \frac{R}{(i-1)(k-iR)} &\approx \rho(i) + \tau(i) \end{aligned}$$

Finally, when L = R, we expect all the unprocessed symbols to be covered. To ensure this, we need that the encoding symbols released then (of degree k/R), must cover the R message symbols. Thus the number of encoding symbols must be $R\log(R/\delta)$, based on the analysis of $\rho(1) = 1$. Straightforward analysis shows that $O(k\log(k/\delta)$ steps are required for decoding and $k + O(\sqrt{k}\log^2(k/\delta))$ encoding symbols have to be received. The probability of decoding failure is less than δ.

6 Conclusions

In this summary we studied two approximations to the ideal digital fountain protocol. Tornado codes have a decoding time linear in the input size, n but are block codes and are not suitable for multicasting to a heterogenous class of receivers. On the other hand, LT codes are rateless codes, but they need decoding and encoding complexity of the order of $O(k\log(k))$. Next week we will study raptor codes that maintain the advantages of LT codes and yet have a much better complexity.

References

[1] P. Elias, "Coding for two noisy channels," *Information Theory, Academic Press*, 1956, pp. 61-74.

[2] J.W.Byers, M.Luby, M.Mitzenmacher, A.Rege. "A Digital Fountain Approach to Reliable Distribution of Bulk Data", *SIGCOMM*. 1998, pp. 56-67

[3] M.Luby and M.Mitzenmacher,A. Shokrollahi,D.Spielman, V. Stemann, "Practical loss-resilient codes",1997

[4] M.Luby and M.Mitzenmacher,A. Shokrollahi,D.Spielman,"Efficient Erasure Correcting Codes",*IEEE Transactions on Information Theory*, vol. 47, issue 2, pp. 569-584, February 2001

[5] M.Luby, "LT Codes", *43rd Annual IEEE Symposium on Foundations of Computer Science*, 2002

[6] David Forney Jr. , "Unpublished Course notes", http://ssg.mit.edu/6.451 , 2003