Tornado Codes

Scribe Notes for 15-853*

Sonesh Surana

1 Introduction

Like linear and Reed-Solomon(RS) codes, Tornado codes are a type of Forward Error Correcting (FEC) code. They were designed by Michael Luby, Michael Mitzenmacher, Amin Shokrollahi, *et al.* (Digital Fountain) to correct erasures with high probability. The encoding and decoding algorithms for these codes have fast and simple software implementations. Because erasures are not only more common than errors but also easily detectable in packet sequences (since packets are indexed) and because of the fast encoding and decoding algorithms, a very attractive application for Tornado Codes is reliable multicast of streaming media over the Internet, where lossy channels are common.

A standard solution to the problem of packet loss over the Internet is retransmission of data. But for real-time transmission this solution can cause unacceptable delays as a result of multiple rounds of communication between the sender and receiver. In such cases FEC codes can be used to reduce the communication overhead between the sender and the receiver. Consider an application that sends a real-time stream of data symbols that is partitioned and transmitted in logical units of blocks. Suppose the network experiences transient and unpredictable losses of at most a p fraction of the symbols in each block where n is the block length. If, instead of sending blocks of n data symbols each, we place $(1-p)n$ data symbols in each block by either selecting the most important parts from the original data stream and omitting the remainder, or by generating a lower quality stream at a fraction of $(1-p)$ of the original stream and filling out the block to its original length n by adding pn check (redundant) symbols, then such a scheme would provide optimal loss protection if the $(1-p)n$ original data symbols can be recovered from *any* set of $(1-p)n$ symbols received. Tornado codes are based on such a strategy.

The design of Tornado codes requires the construction of a random irregular bipartite graph with a carefully chosen degree sequence. The encoding and decoding

*Based on lecture by Professor Bruce Maggs and on [1]

Code	Encoding Time	Decoding Time
Linear	$O(n^2)$	$O(n^3)$
RS	$O(n \log n)$	$O(n^2)$
Tornado	$O(n \log \frac{1}{\epsilon})$	$O(n \log \frac{1}{\epsilon})$

Table 1: Running Time Comparison

algorithms are very similar. Each requires an $\oplus$ (XOR) operation per edge of the graph. The graph is chosen to be sparse so that encoding and decoding are fast. As we will see, the decoding algorithm can be modeled as set of differential equations. The solution to these equations can then be expressed as a polynomial in one variable whose coefficients determine the degree sequence of the graph.

2 Loss Model

The loss model used in Tornado codes is based on the erasure channel described by Elias [2]. According to this model, each symbol is lost with a fixed constant probability p independent of other symbols. In practice, however, losses on the Internet tend to be bursty and correlated. For instance, when a router's buffers fill up, packets start getting dropped, and when the congestion clears, packets stop getting dropped. It seems that the assumption of independent losses is incorrect. But losses on the Internet are generally not related to the actual contents of each packet. Thus if we randomly permute the encoding of symbols in each block before transmission, then even in the face of bursty losses, we can expect that symbols will be lost independent of other symbols. The rate R of a code is defined to be

$$R = \frac{\#\, of\ message(data)\ symbols}{total\ \#\, of\ symbols} \tag{1}$$

In the case of a systematic code with $(1-p)n$ message symbols and pn check symbols forming n total symbols per block, the rate R is

$$R = \frac{(1-p)n}{n} = 1 - p. \tag{2}$$

Elias [2] showed that any random linear code can be used to transmit over the erasure channel at any rate $R < 1 - p$. For linear and RS codes, $R = 1 - p - \epsilon$. Since our loss model is random there is a chance that we might get unlucky and have more errors than expected. The ϵ gives us breathing room to tolerate small deviations in the number of errors. For Tornado codes, the rate R is $R = 1 - p(1 + \epsilon)$ which is not as good as that of linear and RS codes. But the big win is that encoding and decoding are fast (**Table 1**). The decoding algorithm can recover from the random loss of a p fraction of the transmitted symbols in time proportional to $n \log \frac{1}{\epsilon}$ where n is the length of the encoding.

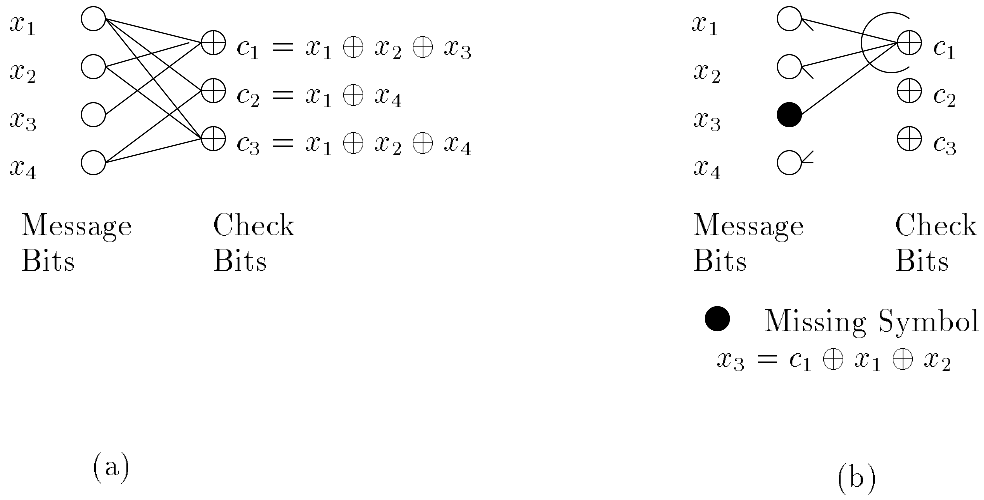

Figure 1: (a)Encoding Step (b)Decoding Step

3 Encoding and Decoding: The Basic Idea

Each symbol can be either a bit or a packet. Without loss of generality assume each symbol is a bit. The message symbols can be chosen freely. The check symbols are computed from the message symbols. The only operation on the bits is $\oplus$. So in a code of block length n and rate R, the encoder takes as input Rn message symbols, generates $(1 - R)n$ check symbols, and transmits a total of n symbols.

Suppose we have a message that is m bits long: $x_1, x_2, x_3, ..., x_m$. To encode, we define a code $\mathcal{C}(B)$ that takes m message bits and generates βm check bits based on a bipartite graph B that defines a mapping from the message bits to the check bits. Each check bit is the $\oplus$ of its neighboring message bits in B(**Fig 1a**). Encoding time is therefore proportional to the number of edges, since we perform an $\oplus$ for each edge in B.

At each decoding step we look for a check bit with all but one of its neighbors (message bits) known. This missing message bit can be set to be the $\oplus$ of the check bit and its known message bits (**Fig 1b**). The decoding time is at most proportional to the number of edges in the graph. The main issue is the construction of a sparse random bipartite such that repetition of the decoding step guarantees (with high probability) the recovery of all message bits if at most $(1 - \epsilon)\beta m$ of the bits have been lost.

If all the check bit values are known, with high probability all the message bits can be recovered. The proof of this claim is not easy and was deferred in class. Now what if we have some missing check bits? We can combat this by cascading multiple levels of bipartite graphs (**Fig 2**). We first use code $\mathcal{C}(B_0)$ to generate βm check bits for the original m message bits, then we use a similar code $\mathcal{C}(B_1)$ to produce $\beta^2 m$ additional check bits for the first set of check bits, and so on. At the last level, we use

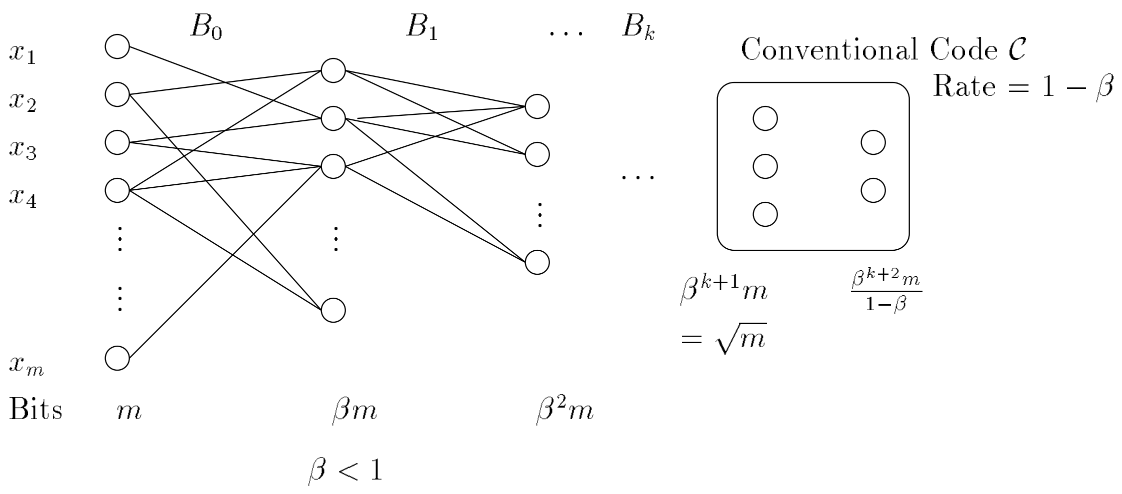

Figure 2: Encoding Structure

a conventional loss resilient code C. In effect, we create a code $\mathcal{C}(B_0, \dots, B_k)$ from a sequence $B_0, B_1, ..., B_k$ of graphs such that B_i has $\beta^i m$ nodes on the left and $\beta^{i+1}m$ nodes on the right. k is selected so that $\beta^{k+1}m$ is approximately $\sqrt{m}$. We end the cascade with a conventional loss-resilient code C of rate $1 - \beta$ with $\beta^{k+1}m$ message bits and $\frac{\beta^{k+2}m}{1-\beta}$ check bits. Thus we get a code $\mathcal{C}(B_0, \dots, B_k, C)$ with n message bits and

$$\sum_{i=1}^{k+1} \beta^i m + \frac{\beta^{k+2}}{1-\beta} = \frac{m\beta}{1-\beta} \tag{3}$$

!lacheck bits. We see that

$$R_{overall} = \frac{m}{m + \frac{m\beta}{1-\beta}} = 1 - \beta \tag{4}$$

If the conventional code C decodes in quadratic time, then decoding the conventional code requires time $O(\sqrt{m}^2) = O(m)$, which is linear in terms of the original number of message bits. The number of edges in the levels preceding the conventional code decreases geometrical with level number. Hence, the total number of edges in all of these levels, and hence the total decode time for these levels is also $O(m)$.

To decode we work backwards from the last level in the cascade. If C recovers all the losses then all the check bits produced by B_k are known, which we can use to recover all the check bits produced by B_{k-1}. In this fashion we can work up backwards till the first level to recover all the check bits produced by B_0, from which we can recover all m original message bits. If we can show that C can recover from a random loss of $\beta(1-\epsilon)$ fraction of its bits and that every code C_i can recover the random loss of $\beta(1-\epsilon)$ fraction of its bits with high probability, we will have shown that $C(B_0, B_1, ..., B_k, C)$ is a rate $1-\beta$ code that can recover from a $\beta(1-\epsilon)$ loss given all the check bits.

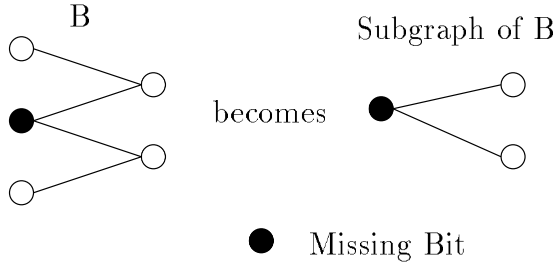

Figure 3: Subgraph of B

4 The Decoding Process

To analyze the decoding process we use a simplified graph model (**Fig 3**). Look at one bipartite graph B in the multiple level cascade, and consider the subgraph of B that consists of all nodes on the left that were lost but have not been decoded thus far, all the nodes on the right, and all the edges between these nodes. Recall that the decoding process requires finding a check bit on the right such that only one adjacent message bit is missing. In terms of the subgraph, this is equivalent to finding a node of degree one on the right, and removing it, its neighbor, and all edges adjacent to its neighbor from the subgraph. This entire sequence of events is one step of the decoding process. This step is repeated until there are no nodes of degree 1 available. The entire process is successful if it does not halt until all nodes on the left are removed, or equivalently, until all edges are removed.

The graphs that are chosen are determined from chosen degree sequences on sparse bipartite graphs. Edges that are adjacent to a node of degree i on the left(right) are refered to as *edges of degree* i on the left(right). Each of the degree sequences is specified by a pair of vectors $(\lambda_1, \lambda_2, \ldots)$ and $(\rho_1, \rho_2, \ldots)$. λ_i is the initial fraction of edges of degree i on the left. ρ_j is the initial fraction of edges of degree j on

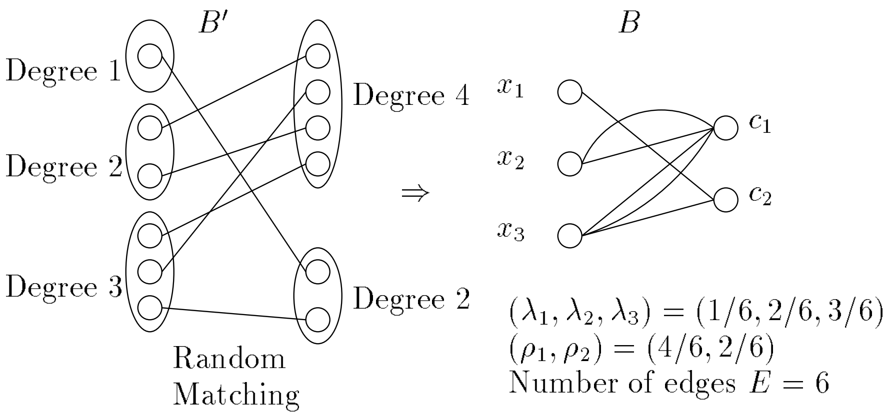

Figure 4: Construction of B given degree sequences (λ), (ρ), and number of edges E

the right. To choose an appropriate random bipartite graph B with E edges and m left nodes and βm right nodes, begin with bipartite graph B' with E nodes each on the left and on the right. Then form each node of degree i on the left side of B by grouping together i nodes on the left side of B', in such a way that every node B' appears in exactly one group. Form right side of B in a similar fashion. Then choose a random matching (i.e., a random permutation) between the two sets of E nodes on B'. This induces a random bipartite graph on B (possibly with multi-edges) with the desired degree structure (**Fig 4**). An important observation to make is that in the corresponding subgraph B' remaining after each decoding step, the matching on B' still corresponds to a random permutation, provided that when a node of degree 1 on the right side of B is selected for elimination, no other information about the structure of the graph is examined (e.g., suppose that a node of degree 1 is selected at random). Hence, the evolution of the degree sequence is a Markov process.

5 Using Differential Equations to Model the Process

The initial random graph is B with m nodes on the left and βm nodes on the right. (λ_i) and (ρ_i) are two vectors such that λ_i and ρ_i are the fractions of edges of degree i on the left and on the right respectively. Let a_l be the average node degree on the left and a_r be the average node degree on the right. On the left,

$$m = \sum_i \frac{\lambda_i E}{i} \tag{5}$$

and

$$a_l = \frac{E}{m} \tag{6}$$

Therefore

$$\frac{1}{a_l} = \frac{m}{E} = \sum_i \frac{\lambda_i}{i} \tag{7}$$

Similarly on the right,

$$\beta m = \sum_i \frac{\rho_i E}{i} \tag{8}$$

and

$$a_r = \frac{E}{\beta m} \tag{9}$$

So

$$\frac{1}{a_r} = \frac{\beta m}{E} = \sum_i \frac{\rho_i}{i} \tag{10}$$

(a) Degree i edge becomes degree $(i-1)$ edge.

(b) Loses j edges of degree j and gains $(j-1)$ edges of degree $(j-1)$.

$\Rightarrow$ edge deleted

Figure 5: (a)Left side (b)Right side

Despite the fact that the decoding process is discrete, we are going to model the elimination process using differential equations in the variable t, time, where time is continuous. We begin by scaling time so that $\Delta t = \frac{1}{E}$ corresponds to one step of the decoding process. Let δ be the fraction of losses in the message. Initially, we can expect that each node on the left is removed with probability $1-\delta$ (because this symbol is received without error and is therefore removed from the subgraph). So the initial subgraph of B contains δm nodes on the left. If the decoding process completes successfully , it runs until time $T = \delta m \Delta t = \frac{\delta m}{E} = \frac{\delta}{a_l}$.
Let
$l_i(t)$ = fraction of edges (in terms of E) with left degree i remaining at time t
$r_i(t)$ = fraction of edges (in terms of E) with right degree i remaining at time t
$e(t)$ = fraction of edges (in terms of E) remaining at time t
Note

$$e(t) = \sum_i l_i(t) = \sum_i r_i(t) \tag{11}$$

At each decoding step, a random node of degree one on the right is chosen and the corresponding node on the left and all of its adjacent edges are deleted. The probability that the edge adjacent to the node of degree one on the right has degree i on the left is $\frac{l_i(t)}{e(t)}$. In this case we lose i edges of degree i (**Fig 5a**) on the left. The difference equation for the *expected change in number of edges* $\mathcal{L}_i(t)$ of degree i on the left can be expressed as

$$\mathcal{L}_i(t+\Delta t) - \mathcal{L}_i(t) = -\frac{i l_i(t)}{e(t)} \tag{12}$$

We know that $l_i(t) = \frac{\mathcal{L}_i(t)}{E} = \mathcal{L}_i(t)\Delta t$. As $E \to \infty$,

$$\frac{dl_i(t)}{dt} = -\frac{i l_i(t)}{e(t)} \tag{13}$$

When we remove a node of degree i on the left, we remove the one edge of degree one from the right, along with the other $i-1$ edges adjacent to this node. The expected number of other edges deleted is $a(t)-1$ where $a(t)=\sum \frac{il_i(t)}{e(t)}$. The right endpoints of these $i-1$ other edges are randomly distributed. If one of the edges is of degree j on the right, we lose j edges of degree j and gain $j-1$ edges of degree $j-1$ (**Fig 5b**). The probability that an edge has degree j on the right is $\frac{r_j(t)}{e(t)}$. For $i>1$ we get the following difference equation:

$$\mathcal{R}_i(t+\Delta t)-\mathcal{R}_i(t)=(r_{i+1}(t)-r_i(t))\frac{i(a(t)-1)}{e(t)} \tag{14}$$

for the *expected change in the number of edges* $R_i(t)$ of degree i on the right where $a(t)=\sum \frac{il_i(t)}{e(t)}$. The corresponding differential equation for $r_i(t)$ is

$$\frac{dr_i(t)}{dt}=(r_{i+1}(t)-r_i(t))\frac{i(a(t)-1)}{e(t)} \tag{15}$$

The case where $i=1$ is special because at each step an edge of degree 1 on the right is removed. So the differential equation for $r_1(t)$ is

$$\frac{dr_1(t)}{dt}=(r_2(t)-r_1(t))\frac{(a(t)-1)}{e(t)}-1 \tag{16}$$

As long as $r_1(t)>0$ there is a node of degree one on the right and the decoding process continues. When $r_1(t)=0$, the process stops. So we want $r_1(t)>0$ until all the nodes on the left are deleted and the process terminates successfully. To solve these differential equations define x so that $\frac{dx}{x}=\frac{dt}{e(t)}$ Then x in terms of t is

$$x=e^{-\int_0^t \frac{d\tau}{e(\tau)}} \tag{17}$$

Substitute $\frac{dx}{x}$ for $\frac{dt}{e(t)}$. Equation (13) becomes

$$\frac{dl_i(x)}{dx}=-i\frac{l_i(x)}{x} \tag{18}$$

Upon integration we get

$$\begin{aligned}\int \frac{dl_i(x)}{l_i(x)} &= \int -i\frac{dx}{x}\\ l_i(x) &= c_i x^i\end{aligned}$$

Note that $x=1$ when $t=0$ from (17) and $l_i(t=0)=\delta\lambda_i$ just before the decoding process starts. So $c_i=\delta\lambda_i$. Therefore

$$l_i(x)=\delta\lambda_i x^i \tag{19}$$

The general solution for $r_i(x)$ is not easy and can be found in the appendix of [1]. The specific solution for $r_1(x)$ can be expressed using the degree sequence functions

$$\lambda(x) = \sum_{i\geq 1} \lambda_i x^{i-1} \tag{20}$$

$$\tag{21}$$

$$\rho(x) = \sum_{i\geq 1} \rho_i x^{i-1} \tag{22}$$

Lemma 1 *The solution $r_1(x)$ of the differential equation (16) is given by*

$$r_1(x) = \delta\lambda(x)[x - 1 + \rho(1 - \delta\lambda(x))] \tag{23}$$

From (23) and the fact that we need $r_1(x) > 0$ until all the left nodes are deleted we can conclude that the condition

$$\rho(1 - \delta\lambda(x)) > 1 - x\,, \quad x \in (0, 1] \tag{24}$$

is required.

6 The Codes

This section presents the codes that transmit at rates arbitrarily close to the capacity of the erasure channel (as $\delta \to p$). Let B be a bipartite graph with n left nodes and βn right nodes. The degree sequence functions λ and ρ chosen should satisfy condition (23). Let d be a positive integer that is used to trade off the average degree with how close we can make δ to $\beta = 1 - R$ and finish successfully with high probability.

Let $H(d) = \sum_{i=1}^{d} \frac{1}{i}$ be the the harmonic sum truncated at d. $H(d) \backsim \ln d$. For all $i = 2, \ldots, d+1$ the fraction of edges of degree i on the left is

$$\lambda_i = \frac{1}{H(d)(i-1)} \tag{25}$$

The average left degree $a_l = \frac{H(d)(d+1)}{d}$. Recall from (6) and (9) that $a_r = \frac{a_l}{\beta}$. The right degree sequence is defined by the Poisson distribution with mean a_r. For all $i \geq 1$ the fraction of edges of degree i on the right equals

$$\rho_i = \frac{e^{-\alpha}\alpha^{i-1}}{(i-1)!} \tag{26}$$

where α is chosen to guarantee that the average degree on the right is a_r. In other words, α satisifies $\frac{\alpha e^{\alpha}}{e^{\alpha}-1} = a_r$. When $\delta = \frac{\beta}{1+\frac{1}{d}}$ in (24), the condition is satisfied.

References

[1] M.Luby, M.Mitzenmacher, M.A.Shokrollahi *et al.*: *Practical Loss-Resilient Codes.*

[2] P.Elias: *Coding for Two Noisy Channels*, Information Theory, Third London Symposium, September 1955.