

# Fountain Codes and their possible application in standards like GSM

T. L. Grobler, J. C. Olivier and J.D. Vlok

**Abstract—** In this paper we review the concept of a fountain code and explore its possible application in standards like GSM. Our main focus is on the decoding of Tornado codes (predecessor of fountain codes) in typical mobile wireless channels encountered in GSM and WiMax standards. Channel conditions considered include AWGN and flat fading. The application of fountain codes in GSM is appealing due to the graph structure of these codes and the coding configuration in current GSM standards.

**Index Terms—**Fountain codes, Tornado, LT, Raptor, belief propagation, factor graph, GSM.

## I. INTRODUCTION

Digital fountain codes use the concept of a sparse-graph at their core. The first sparse-graph code was introduced by R.G. Gallager in [1,2] in the early 1960's and is known as LDPC (Low-Density Parity-Check) codes.

One of the code families derived from the sparse-graph concept was digital fountain codes. Digital fountain codes are designed for a binary erasure channel, in which each codeword symbol is lost with a fixed constant probability  $p$  in transit independent of all the other symbols (an example is the internet).

The digital fountain concept [3] started as a data carousel (broadcast disk) [4]. In this approach the source loops through all transmission packets continuously; the receivers may log onto this stream at any time and download packets until they have received the entire message. A data carousel can be seen as an imperfect approximation of an ideal solution, which one can call a digital fountain. The difference between a data carousel and a digital fountain is that a digital fountain can reconstruct the message from any subset of encoding packets equal in length to the original message (note that the packets are encoded using some sort of FEC), while the decoding properties of a data carousel is not as strict (the packets used does not have to be any subset or a specific length). A digital fountain can be compared to a running tap. When you fill your cup you do not care which droplets land in the cup, but only that you require enough water to quench your thirst. The above metaphor also highlights another important property of digital

fountain codes namely it's ability to generate an infinite amount of encoded packets from the original source.

One of the first coding schemes chosen to create digital fountains is Reed-Solomon (RS) codes [5]. The reason RS codes were considered is the fact that the original message can be constructed from  $k$  RS symbols. Where  $k$  represents the original message size (in packets) and  $n$  represents the code size (in packets).

The two main drawbacks of RS codes are the limited amount of distinct decoding symbols and the quadratic decoding algorithm used by it [3,6].

The next coding scheme proposed to approximate a digital fountain is a sparse-graph erasure code called Tornado codes [7,8]. This sparse property of Tornado codes solved the quadratic time problem of RS codes (made it linear). This does come with a price; the receivers have to receive a little bit more than  $k$  packets to correctly decode the original message. Take note that a Tornado code is not a true fountain code (rather a predecessor), due to the fact that it can not produce an unlimited supply of unique code bits.

The next logical evolution was to develop a rateless code which is a true fountain code. LT (Luby Transform) codes were the first rateless codes and were discovered by Luby [9]. LT codes have a similar graph structure to Tornado codes, though its graph is not predefined giving LT codes their rateless property. To make this structure possible each encoded symbol must store a list of its neighbors in real time, which can be accomplished in practice if the receiver and transmitter share the same random seed and degree distribution. They can agree on this before transmission begins. The underlying degree distribution is a very important design choice and the robust soliton distribution is a popular choice [9,10]. Using this degree distribution leads to encoding symbols with average degree  $O(\ln k)$ . Encoding symbols with average degree  $O(\ln k)$  leads to codes that can encode in time proportional to  $\ln k$  and decode in time proportional to  $k \ln k$  [9]. These codes also require that the receiver receive a little more than  $k$  packets before accurate decoding can be accomplished. A detailed comparison of the implementation of RS, Tornado and LT codes as digital fountains can be found in [3,10].

The only improvement on LT codes is to lower the average degree to a constant which would lead to decoding in time proportional to  $O(k)$ . Raptor codes [11,12,13] accomplish just this by using pre-coding. An average degree distribution of  $O(\ln k)$  is required to cover each message node with high probability. To remove this restriction we can pre-code a message using an erasure code like Tornado. If we treat the encoded message  $M'$  as the

message we do not need to recover every packet of  $M'$  in order to recover  $M$  (original message), but instead just a constant fraction of the packets of  $M'$ . This reduces the decoding time complexity to  $O(k)$ . Some research on using digital fountain codes on non-erasure channels are done in [13].

The main aim of the paper is to kick off research exploring the use of fountain codes in standards like GSM. The best place to start is to investigate the decoding of Tornado codes on non-erasure channels like AWGN with flat fading added (corruption, not erasure channel). This is done due to the fact that the decoding of Tornado codes can easily be extended to other fountain codes (having a similar graph structure). The next section introduces the basics of Tornado code structure and design (for an erasure channel). Section III explains how a Tornado code can be decoded on an AWGN channel using iterative belief propagation. The results of this approach are given in section IV and are followed by a discussion of these results. Before the conclusion a short theoretical description of how fountain codes could be used in GSM is presented.

## II. TORNADO CODES

### A. Basic structure

The basic structure of a Tornado code consists of layered bipartite graphs. Each bipartite graph  $B$  is associated with a code  $C(B)$  with  $k$  message bits and  $\beta k$  redundant bits (check bits), where  $0 < \beta < 1$ . Each graph  $B$  thus consists of  $k$  left nodes and  $\beta k$  right nodes.

The encoding of  $C(B)$  is accomplished by setting each check bit equal to the XOR of its neighboring message bits. For the layered graph the above procedure is continued for each layer. The codes used are systematic and sparse.

The decoding of a layer on an erasure channel is accomplished by substituting the correctly received bits into the graph structure and then solving the unknown bits iteratively. This procedure only works if enough bits are received correctly. The above is explained in more detail in [7].

Now that a single bipartite graph was introduced it can be cascaded to form a Tornado code. One first uses  $C(B)$  to produce  $\beta k$  check bits for the original  $k$  message bits. One then uses a similar code to produce  $\beta^2 k$  check bits for the  $\beta k$  check bits of  $C(B)$ , and so on. The last level of a Tornado code may use a conventional erasure correcting code like RS (Reed-Solomon) [5]. Implementing Tornado codes on non-erasure channels using belief propagation prohibits this approach.

For this article the last layer consists of a specialized bipartite graph specializing in maximizing error recovering of left as well as right hand side nodes for a specific layer. The other layers are only designed to maximize error recovery of left hand nodes. The reason being that these layers assume that the conventional error correcting code has performed all the necessary corrections of the left nodes, at the second last layer.

The Tornado graph structure can now be formally defined as follow [7]. A family of codes  $C(B_0), \dots, C(B_m)$

can be constructed from a family of graphs  $B_0, \dots, B_m$ , where  $B_i$  has  $\beta^i k$  left nodes and  $\beta^{i+1} k$  right nodes. The variable  $m$  is chosen that  $\beta^{m+1} k$  is roughly  $\sqrt{k}$ . The cascade ends with an erasure correcting code  $C$  of rate  $1 - \beta$  with  $\beta^{m+1} k$  message bits. The code  $C(B_0, B_1, \dots, B_m, C)$  has  $k$  message bits and

$$\sum_{i=1}^{m+1} \beta^i k + \beta^{m+2} k / (1 - \beta) = k\beta(1 - \beta) \quad (1)$$

check bits formed by using  $C(B_0)$  to produce  $\beta k$  for the  $k$  message bits, using  $C(B_i)$  to form  $\beta^{i+1} k$  check bits for the  $\beta^i k$  bits produced by  $C(B_{i-1})$ . And finally using  $C$  to produce an additional  $k\beta^{m+2} / (1 - \beta)$  check bits for the  $\beta^{m+1} k$  bits output by  $C(B_m)$ . As  $C(B_0, B_1, \dots, B_m, C)$  has  $k$  message bits and  $k\beta / (1 - \beta)$  check bits, it is a code of rate  $1 - \beta$ .

The decoding of a complete Tornado code on an erasure channel is done as follows. The conventional erasure code recovers all the missing left and right check nodes of the last layer, making all the check bits of  $C(B_m)$  known. This can now be used to correct the erasures in the inputs of  $C(B_m)$ . This continues upwards until the original  $k$  message bits can be recovered.

### B. Degree sequences

An edge is a line that connects any two nodes, one from the left with one on the right. One refers to edges that are adjacent to a node of degree  $i$  on the left (or right) as edges of degree  $i$  on the left (or right), where the degree of a node is determined by the amount of edges connected to it. Each of the degree sequences is specified by a pair of vectors  $(\lambda_1, \dots, \lambda_m)$  and  $(\rho_1, \dots, \rho_m)$ , where  $\lambda_i$  is the initial fraction of edges on the left of degree  $i$  and  $\rho_j$  is the initial fraction of edges on the right of degree  $j$ . Note that the graphs are specified in terms of fractions of edges, and not nodes, of each degree. The sequence  $\lambda$  give rise to a generating polynomials  $\lambda(x) = \sum_i \lambda_i x^{i-1}$ . The average left degree of the graph is thus equal to

$$a_\ell = \left[ \sum_i \lambda_i / i \right]^{-1} \quad (2)$$

If  $E$  is the number of edges in the graph, then the number of left nodes of degree  $i$  is

$$E\lambda_i / i \quad (3)$$

and hence the number of left nodes is

$$E \sum_i \lambda_i / i \quad (4)$$

The above can be repeated for the right side nodes, as long as  $E$  remains constant [7].

For an effective erasure correcting code the degree sequences have to satisfy the following theorem from [7]

Let  $k$  be an integer, and suppose that  $C = C(B_1, \dots, B_m, C)$  is a cascade of bipartite graphs as explained in the previous section, where  $B_1$  has  $k$  left nodes. Suppose that each  $B_i$  is chosen at random with edge

degrees specified by  $\lambda(x)$  and  $\rho(x)$ , such that  $\lambda(x)$  has  $\lambda_1 = \lambda_2 = 0$ , and suppose  $\delta$  is such that

$$\rho(1 - \delta \cdot \lambda(x)) > 1 - x \quad (5)$$

for  $x \in (0,1]$ . Then, if at most a  $\delta$ -fraction of the coordinates of an encoded word in  $C$  are erased independently at random, the erasure decoding algorithm of the previous section terminates successfully with probability  $1 - O(k^{-3/4})$ , and does so in  $O(k)$  steps.

### C. Designing a Tornado code for the erasure channel

The Tornado code consists of multiple layers of bipartite graphs. All of the intermediate layers are designed using the heavy tail distribution for the left nodes (discussed below). The last layer is designed using the double heavy tail distribution for the left nodes (discussed below). This is done due to the fact that the double heavy tail distribution is more effective than the heavy tail distribution when correcting erasures on the right and left nodes of the bipartite graph, according to [7]. The code in this section is designed to satisfy condition (5).

An intermediate layer is designed in the following manner as discussed in [7]. Let  $B$  be an intermediate bipartite graph with  $k$  left nodes and  $\beta k$  right nodes. The left degree sequence is described by the following truncated heavy tail distribution. Let  $H(D) = \sum_{i=1}^D 1/i$  be the truncated harmonic sum truncated at  $D$ , where  $D$  is an integer used to trade off the average degree with how well the decoding process works. The fraction of edges of degree  $i = 2, \dots, D+1$  on the left is given by

$$\lambda_i = 1/(H(D)(i-1)) \quad (6)$$

From equation (2) and (6) it can be shown that  $a_l = H(D)(D+1)/D$ . The average right degree  $a_r$  needs to satisfy  $a_r / \beta$ . 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

$$\rho_i = \frac{e^{-\alpha} \alpha^{i-1}}{(i-1)!} \quad (7)$$

where  $\alpha$  is chosen such that the average degree on the right is equal to  $a_r$ . In other words,  $\alpha$  satisfies

$$\alpha e^\alpha / (e^\alpha - 1) = a_r \quad (8)$$

This approach however does not work due to the fact that there are nodes of degree two on the left. To overcome this problem one can make a small change in the structure of the graph  $B$ . Let  $\gamma = \beta / D^2$ . One can split the  $\beta k$  right nodes of  $B$  into two distinct sets, the first set consisting of  $(\beta - \gamma)k$  nodes and the second set consisting of  $\gamma k$  nodes. The graph  $B$  is then formed by taking the union of the two graphs  $B_1$  and  $B_2$ .  $B_1$  is formed between the  $k$  left nodes and  $(\beta - \gamma)k$  right nodes as described above.  $B_2$  is formed between the  $k$  left nodes and the second set of  $\gamma k$  right nodes, where each of the  $k$  left nodes has degree three and the  $3k$  edges are connected randomly to the  $\gamma k$  right nodes.

For this graph  $\delta = \beta(1 - 1/D)$  and is defined by equation (5).

The last layer is designed using a different technique than the other layers [7]. The edge distribution on the left is now a double heavy tail. In other words  $\lambda(x) = \bar{\lambda}(x^2)$ , where  $\bar{\lambda}$  is the edge distribution function of the heavy tail distribution. The right edge distribution is calculated using linear programming (simplex method). Knowing  $\lambda(x)$  and  $\delta$  (choosing it at least equal to  $\delta = \beta(1 - 1/D)$ ) one can calculate  $\rho(x)$ . The objective is to find  $\rho_m; m \in M$ , where  $M$  is a fixed set of positive integers and has a size of at least  $N$ . Let  $x_i = 1/N$  for  $i = 1, 2, \dots, N$ . The simplex method can now be used to minimize

$$\sum_i (\rho(1 - \delta \lambda(x_i)) + x_i - 1) \quad (9)$$

subject to  $\rho_i > 0$ ,  $\rho(1 - \delta \lambda(x_i)) > 1 - x_i$ ,  $\sum_i \rho_i = 1$  and  $\sum_k \lambda_k / k = \sum_i \rho_i / i$  (if the amount of left and right hand nodes are equal). This solution is only feasible if  $\rho(1 - \delta \lambda(x)) > 1 - x$  for all  $x \in (0,1]$ .

### D. An example design

The first step in designing a Tornado code is choosing the dimensions of the code, and the amount of layers. A  $(n, k) = (80, 40)$  code with 2 layers is considered here. Although a typical Tornado code is at least 10000 bits long, the example considered here is sufficient to highlight a flaw in Tornado codes when used on an AWGN channel with flat fading added. The first layer will consist of the heavy tail distribution as described in the previous section with  $D = 3$  and  $\beta = 0.5$ . The first layer will thus consist of 40 right nodes and 20 left nodes. From section II-C one knows that the first layer consists of the union between two graphs  $B_1$  and  $B_2$ . The nodes on the right needs to be divided into two unique sets. One first needs to calculate  $\gamma = \beta / D^2 = 1/18$ . Now  $B_1$  will use  $\lfloor (\beta - \gamma)k \rfloor = 17$  and  $B_2$  will use 3 nodes on the right. Both graphs will use all of the left hand nodes.

First  $B_1$  is designed and then  $B_2$  is added to form  $B$ . The polynomials  $\lambda(x)$  and  $\rho(x)$  of  $B_1$  is designed by using the values calculated in Table I and can be found in Table II. Note from Table I that a new  $\beta$  is used to calculate the average right degree. With  $\lambda(x)$ ,  $\rho(x)$  and equation (3) the degrees of each node can be determined. Since equation (3) does not divide the edges of  $B_1$  perfectly, seven edges remain (only 92 edges are used). The remaining edges are donated to  $B_2$ . To complete  $B$ , one needs to add  $B_2$  to  $B_1$ . This is easily accomplished by adding three edges to each left node and assigning these edges (including the seven donated ones) randomly to the three nodes of  $B_2$ . One such assignment is shown in Table III.

TABLE I  
VALUES NEEDED FOR THE DESIGN OF  $B_1$

Unknown	Value	Equation
$E_{B_1}$	99	(4) - rounded to

		largest integer
$\beta_{new}$	0.425	17/40
$a_\ell$	2.444	(2)
$a_r$	5.5	$a_r = a_\ell / \beta_{new}$
$\alpha$	5.477	(8)
$\delta$	0.333	$\delta = \beta(1 - 1/D)$

$E = 78$
----------

TABLE V  
COMPLETE DESIGN OF THE SECOND LAYER

Left Nodes		Right Nodes	
Degree	Amount	Degree	Amount
3	13	1	5
5	5	3	1
7	2	5	14

TABLE II  
DEGREE DISTRIBUTION POLYNOMIALS FOR  $B_1$

Unknown	Equation
$\lambda(x)$	(6)
$\lambda(x) = 0.5455x + 0.2727x^2 + 0.1818x^3$	
$\rho(x)$	(7)
$\rho(x) \approx 0.0042 + 0.0229x + 0.0627x^2 + 0.1145x^3 + 0.1568x^4 + 0.1718x^5 + 0.1568x^6 + 0.1227x^7 + 0.084x^8$	
$E_{B_{1,new}} = 92$	$E_{B_2} = 127$
$E = 219$	

TABLE III  
COMPLETE DESIGN OF THE FIRST LAYER

Left Nodes		Right Nodes	
Degree	Amount	Degree	Amount
5	26	2	1
6	9	3	2
7	5	4	3
		5	3
		6	3
		7	2
		8	2
		9	1
		35	1 (B2)
		46	2 (B2)

The last layer of the graph is designed by using linear programming. It consists of the 20 parity bits of the first layer on the left and 20 parity bits on the right. Assuming  $D$  is also equal to three for the last layer and that  $\delta$  remains the same as the previous layer one can calculate  $\lambda(x)$  and  $\rho(x)$  as shown in Table IV. With  $\lambda(x)$  and equation (4) one can calculate  $E$ , which is equal to 78 in this case. With  $E$ ,  $\lambda(x)$ ,  $\rho(x)$  and equation (3) one can construct Table V. The reason for a node of degree 3 on the right is that two edges remained after dividing the edges on the right. These two edges were assigned to a random node of degree one. A random (80, 40) Tornado code is shown in Fig. 1.

TABLE IV  
DEGREE DISTRIBUTION POLYNOMIALS FOR SECOND LAYER

Unknown	Equation
$\lambda(x)$	$\lambda(x) = \bar{\lambda}(x^2)$
$\lambda(x) = 0.5455x^2 + 0.2727x^4 + 0.1818x^6$	
$\rho(x)$	Linear Programming
$\rho(x) = 0.079 + 0.9921x^4$	

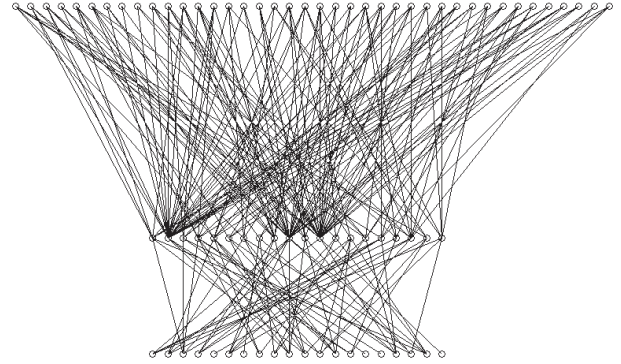


Fig. 1. A random (80, 40) Tornado code

### III. DECODING USING BELIEF PROPOGATION

Since a Tornado graph can be represented as a factor graph, in theory it should be possible to implement iterative belief propagation on such a code. Tornado codes can be represented as a factor graph by inserting the hidden check nodes into Fig. 1. In a factor graph the messages can be passed as LLR's (log likelihood ratios), which is defined as

$$\Lambda(m_{a,b}) = \ln \left[ \frac{p(m_{a,b} = +1)}{p(m_{a,b} = -1)} \right] \quad (10)$$

where  $m_{a,b}$  is the message passed from the a-layer to the b-layer.

A factor graph usually consists of two types of nodes; variable nodes and check nodes. A variable node usually represents a received bit and is assigned a channel LLR, defined as:

$$\Lambda(m_n) = \frac{4}{N_0} \alpha_n r_n \quad (11)$$

where  $r_n$  is the soft value of the  $n^{\text{th}}$  received bit,  $\alpha_n$  is the average fading amplitude of  $r_n$  and is the single-sided noise power spectral density.

A check node forces all the variable nodes connected to it to be even. Each of these types of nodes is updated using different formulas. The input LLR's of a variable node are summed together to produce the output branch LLR (including channel LLR), and is described as:

$$\Lambda(m_o) = \sum_{i \neq o} \Lambda(m_i) + \Lambda(m_n) \quad (12)$$

For a check node

$$\Lambda(m_o) = 2 \cdot \tanh^{-1} \left[ \prod_{i \neq o} \tanh \left( \frac{\Lambda(m_i)}{2} \right) \right] \quad (13)$$

is used. To update a node each branch of a node needs to be updated with the above formulas. All branches are usually set to zero for the first iteration.

The order in which one updates the nodes can be chosen uniquely for each factor graph. Standard LDPC iterative belief propagation involves updating the variable nodes first and then the check nodes. Take note that some sequences may lead to bad decoding. After such a sequence has been completed, one can stop or continue with another cycle. Usually a sequence is repeated until a codeword (all the checks are satisfied) or a certain predetermined amount of cycles are reached.

Decoding of a variable node is done by calculating the intermediate value

$$\Lambda(m_v) = \sum_i \Lambda(m_i) + \Lambda(m_n) \quad (14)$$

and performing hard decisions on the value. A similar algorithm is discussed in [15].

#### IV. RESULTS

Three codes were tested on an AWGN channel with flat fading and the resulting BER graphs are shown below. The details of the different codes can be found in section V. The channel was constructed as in [14] and QPSK was used as modulation technique. For fading Rician factors of -100dB and 9 dB were used and Doppler frequencies of 100Hz and 33Hz. All of the codes were decoded using 10 iterations of standard LDPC iterative belief propagation, as discussed in section III.

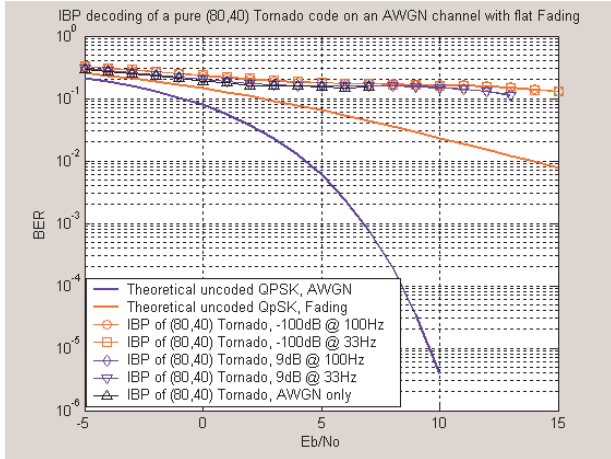


Fig. 2. BER curve of the pure Tornado (80,40)

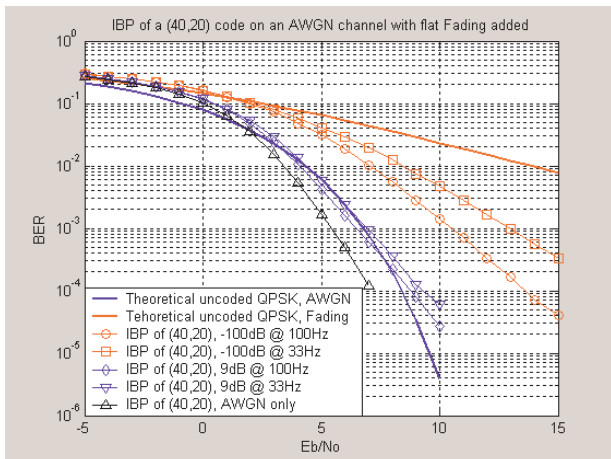


Fig. 3. BER curve for the (40,20) code

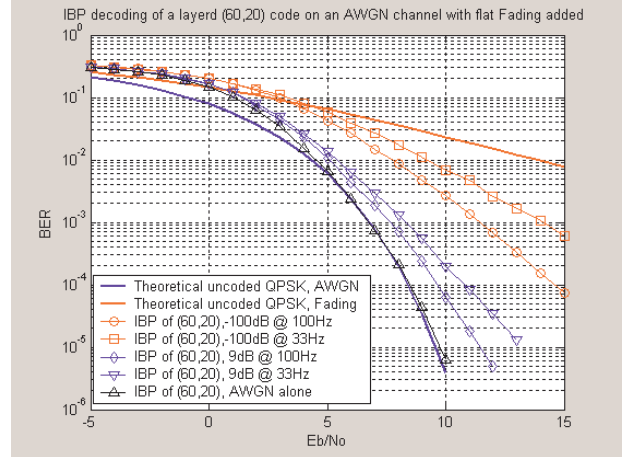


Fig. 4. BER curve for the layer (60,20) code

#### V. INTERPRETING THE RESULTS

The pure (80,40) Tornado code of section II-D performs very bad on an AWGN channel with flat fading. The main reason for such bad performance can be attributed to the addition of  $B_2$  to  $B_1$  to create the first layer of the Tornado code. Because the edges introduced by  $B_2$  are assigned to only a handful of the right hand side nodes, a lot of parallel branches are created. These two cycle loops are very bad for belief propagation and causes the bad BER curves of Fig 2. This is however only one of the possible explanations for this codes bad performance, which can include other factors like four-cycle loops and bad distribution. When only the second layer of the pure Tornado (80,40) code is used a new (40,20) code is created. This (40,20) code performs a lot better since the two cycle loops of the first layer is not present. One can also construct a (60,20) code by using the last layer of a pure (80,40) Tornado code for both layers of a new code. This code performs worse than the single layer (40,20) code. This can be attributed to a flaw in the structure of a Tornado bipartite graph. To obtain the same  $E_b/N_0$  for different codes a certain amount of noise must be added for each code, which is dependent on the code rate. For a (40,20) and a (60,20) code to have the same  $E_b/N_0$  value one needs to add 1.5 times more noise to each bit of the (60,20) code than for the (40,20) code. Now the only effect of the second layer on the first layer is that the 20 edges going into the xor's of the first layer are more accurate. These accurate values however can not compensate for the extra amount of noise carried by the message bits. The 20 edges are only a small amount of the information exchanged in the first layer. In other words most of the information used by belief propagation decoding of the first layer is produced by the first layer (not including the 20 edges from the second layer). A Tornado code is thus a code consisting of multiple layers, each layer can decode effectively by itself, but the amount of information distributed between layers is so small that it has a minimal affect.

Another thing to note is that the (40,20) and (60,20) codes perform better than uncoded QPSK when compared under fading conditions although the (60,20) code performs worse under AWGN conditions.

The only possible solution for the flaw in Tornado codes discussed above is to redesign it for an AWGN channel. The layers should be designed using principals in [13]. For the second layer an additional refinement is needed. The second layer should use the first parity layer and the original message bits in its construction. This approach should produce a lot more messages during belief propagation and lead to a better code. This approach should be applied to every layer regardless of its depth.

## VI. POSSIBLE USE OF FOUNTAIN CODES IN GSM

Tornado codes exhibit certain difficulties (including the flaws discussed in section V and the predefined graph structure) for application in GSM, though other fountain codes such as LT and Raptor codes might offer a solution. The difference between LT codes and Tornado codes is that LT codes only use a single layer that is generated during encoding on the fly. All the message bits of an LT code are placed on the left side of the graph. Each check node value is generated by picking a random amount of message bits (according to some distribution) and then calculating the binary sum of these bits. This is repeated for as many check nodes as required. LT codes are non-systematic and so only the check node values are sent over the channel. Raptor codes are almost identical to LT codes except for the inclusion of a pre-code. The message bits in a Raptor code are encoded by an appropriate pre-code and then encoded by a LT code. Since Raptor and LT codes can be represented using factor graphs it can easily be decoded using the principals discussed in section III.

GSM uses a punctured convolutional encoder to encode frames and each frame is tested for validity at the receiver. If an invalid frame is found, it is retransmitted using a different puncturing scheme. This leads to more available data at the receiver for decoding (does not use all the received data). Theoretically, Fountain codes have the ability to utilize all the retransmitted information for decoding. Raptor or LT codes can be used to replace the puncturing scheme, e.g. a message can be encoded using a LT code and if the frame at the decoder is invalid the transmitter retransmits the message using a different LT structure. The only challenge is that the combined LT graph of both transmissions must be an efficient code as well. Now the receiver can use both transmissions for successful decoding. This can be continued until a valid frame is received. In short the LT code used for this approach should adhere to the following

- The LT code itself and the combined LT codes must be an effective code.
- Four cycle loops must be limited.
- Some LDPC design principals may be incorporated.
- The design rules for fountain codes given in [13] must be applied.

## VII. CONCLUSION

A pure Tornado code uses an inefficient code structure and is not suitable for noisy channels. It can be redesigned for noisy channels as discussed in section V. On the other hand pure fountain codes do not share this flaw with Tornado codes. These codes can be used in standards like

GSM due to its ability to generate an infinite amount of code bits. This makes it possible (in theory) for the receiver to use all the retransmitted bits for decoding while the GSM decoder can only use some of the retransmitted bits (due to puncturing). All fountain codes can be decoded using the principals discussed in section III on noisy channels. This concept is worth researching further.

## VIII. REFERENCES

- [1] R.G. Gallager, "Low density parity check codes.", *IRE Trans. Info. Theory* IT-8: 21-28, 1962.
- [2] R.G. Gallager, *Low Density Parity Check Codes. Number 21 in MIT Research monograph series.* MIT Press., 1963.
- [3] J. Byers, M. Luby, and M. Mitzenmacher. "A Digital Fountain Approach to Asynchronous Reliable Multicast.", *IEEE Journal on Selected Areas in Communications*, 20(8): 1528-1540, October 2002.
- [4] S. Acharya, M. Franklin, and S. Zdonik, "Dissemination based data delivery using broadcast disks," *IEEE Pers. Commun.*, (2):50-60, Dec 1995.
- [5] I.S. Reed and G. Solomon, "Polynomial codes over certain finite fields.", *Journal of the Society for Industrial and Applied Mathematics*, (8):300-304, June 1960.
- [6] M. Mitzeumacher, "Digital Fountains: A Survey and Look Forward", *ITW, San-Antonio, Texas*, 271-276, October 2004.
- [7] M. Luby, M. Mitzenmacher, A. Shokrollahi, and D. Spielman, "Efficient erasure correcting codes", *IEEE Trans. Inform. Theory*, (47):569-584, Feb. 2001.
- [8] M. Luby, M. Mitzenmacher, and A. Shokrollahi, "Analysis of random processes via and-or tree evaluation," in *Proc. 9th Annu. ACM-SIAM Symp. Discrete Algorithms*, 364-373, Jan. 1998, pp. 364-373.
- [9] M. Luby, "LT codes.", In *Proc. of the 43rd Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pp. 271-282., 2002.
- [10] J. Byers, M. Luby, M. Mitzenmacher, and A. Rege, "A digital fountain approach to reliable distribution of bulk data", in *Proc. ACM SIG-COMM*, Vancouver, BC, Canada, 56-67, Aug. 1998.
- [11] A. Shokrollahi, "Raptor Codes", *IEEE Transactions on Information Theory*, 52(6):2551-2567, June 2006.
- [12] A. Shokrollahi, "Raptor Codes", ISIT, Chigago, USA, 36, June 2004.
- [13] O. Etesami and A. Shokrollahi, "Raptor Codes on Binary Memoryless Symmetric Channels", *IEEE Transactions on Information Theory*, 52(5):2033-2051, May 2006.
- [14] L. Staphorst, "Viterbi decoded linear block codes for narrowband and wideband wireless communication over mobile fading channels," Master's dissertation, University of Pretoria, 2005.
- [15] J.D. Vlok, "Sparse graph codes on a multi-dimensional WCDMA platform," Master's dissertation, University of Pretoria, 2007.

**Trienko L. Grobler** received his B.Eng (Computer Engineering) and B.Eng (Hons) degrees at the University of Pretoria in 2005 and 2006 respectively. He is currently

pursuing the M. Eng degree in telecommunication and signal processing at the same establishment. His research interests mainly include signal processing and cryptography.