Designing Tornado Codes as Hyper Codes for Improved Error Correcting Performance

Kristian Nybom, Jerker Björkqvist
Department of Computer Science, Åbo Akademi University
kristian.nybom@abo.fi, jerker.bjorkqvist@abo.fi

Abstract

Digital services are increasingly being delivered to user terminals over wireless multicast networks. When the target service requires uncorrupted data delivery, there is an increasing need for developing techniques for reliable content delivery. In multicast networks, the typical way of providing reliable content delivery is to apply forward error correcting techniques (FEC). In this paper, we introduce a modification to the Tornado code, where we combine the structures of Tornado codes and Hyper codes. Standard Tornado codes have the drawback that they are dependent on the error distribution on the data path. By combining the Tornado code with a Hyper code, we decrease the probability of the Tornado decoder failing due to inopportune error distribution in the source data.

We demonstrate, based on a simulated transmission channel, that the error correcting performance is improved, at the cost of slightly longer decoding times, by using a combination of Hyper codes and Tornado codes, compared to the standard Tornado codes.

1. Introduction

Forward Error Correction (FEC) mechanisms provide transport protocols with reliable delivery of content. FEC mechanisms are especially suitable for IP multicast protocols when feedback to the transmitter is either costly or impossible. FEC contributes protocols with the ability to overcome both erasures and bit-level corruption. The primary contribution of FEC codes to IP multicast protocols is, however, erasure correction, as the network layers in IP multicast protocols will detect and discard corrupted packets.

In 1998, Byers et al. [1] introduced a new FEC code, called the Tornado code. The same year, Hunt et al. [2] presented a new family of FEC codes, called Hyper codes. This paper shows with simulations that by designing Tornado codes as Hyper codes, the error correcting performance of the Tornado code can be notably improved at the cost of slightly longer decoding times.

2. The Tornado Code

The Tornado code introduced by Byers et al. [1] consists of several layers of bipartite graphs. In the leftmost graph, the left side nodes are called message nodes and correspond to the source symbols. A symbol is some predefined number of bits. All the remaining nodes are called check nodes and correspond to redundancy symbols. An example of a Tornado code structure with three layers of nodes is illustrated in Fig. 1. Between the right side nodes and the left side nodes in each bipartite graph there exist edges, specifying how the right side nodes depend on the left side nodes. The degree of a node is equal to the number of edges connected to the node. The check nodes are calculated as the exclusive-OR of all the left side nodes in the bipartite graph with which the check nodes share edges. Decoding of a block is done using the following algorithm:

"Given the value of a check node and all but one of its message nodes, set the missing node to be the XOR of the check node and all of the check nodes known message nodes."

If a check node is required for decoding, but the value of the node is unknown, a similar decoding algorithm must be performed in the bipartite to the right, so that the check node is recovered.

The error correcting performance of the code is extremely dependent on the design of the bipartite graphs. Luby [3] states that the degree distributions of the left side nodes in each bipartite graph should be equal to the Soliton distribution and the degree distributions of the right side nodes should be equal to the Poisson distribution.

Proceedings of the Advanced International Conference on Telecommunications and
International Conference on Internet and Web Applications and Services (AICT/ICIW 2006)
0-7695-2522-9/06 $20.00 © 2006 IEEE

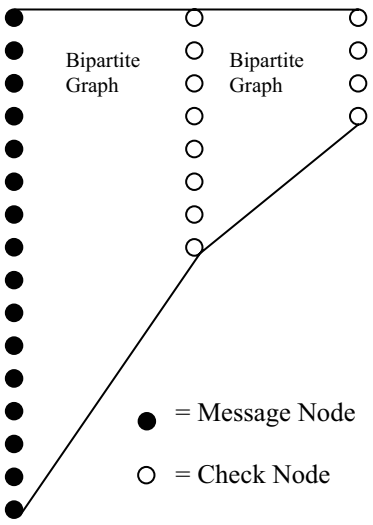


Fig. 1. The structure of a Tornado code with three layers of nodes. The black dots correspond to the source symbols and the white dots correspond to the redundancy symbols.

Luby et al. [4] show that if each bipartite graph, B_i, has $\beta^i K$ left side nodes and $\beta^{i+1} K$ right side nodes, where i=0..m, K is the number of source symbols and β is chosen between zero and one, then the number of bipartite graphs, m+1, should be chosen so that $\beta^{m+1} K$ is roughly $\sqrt{K}$. This yields a code with code rate

$$R = \frac{K}{N} = \frac{K}{\sum_{i=0}^{m+1} \beta^i K} = 1 - \beta \qquad (1)$$

Elias [5] showed that if the capacity of an erasure channel is 1-p, where p is the fixed constant probability that a codeword symbol is lost, then a random linear code of rate R<1-p can be used to transmit over this channel. Therefore, this code is able to recover from a random loss of β fraction of its nodes with high probability.

3. Hyper Codes

Hunt et al. [2] describe a new family of FEC codes, called Hyper codes. These codes can be understood as multi-dimensional codes. Hunt et al. prove that Hyper codes offer good but not exceptional error-rate performance, for a given block size and code rate. Hyper codes have significantly enhanced distance properties compared to codes that are not structured in this manner.

When using Hyper codes, the source bits are arranged in an N-dimensional "box". In every dimension, parity bits are calculated, but the source bits are permuted between the dimensions. The permutation of the source bits increases the minimum distance, which can be seen as an improvement of the error correcting performance of the code. The Hyper code decoder relies on soft-decision decoding.

4. Combination of Hyper Codes and Tornado Codes

By using the methodology of Hyper codes when designing Tornado codes, the error correcting performance of Tornado codes can be improved. The combination of the codes results in the following encoding algorithm:

1. Calculate the redundancy symbols in the first dimension as described in section 2.
2. Permute the message nodes.
3. Calculate new redundancy symbols using the same graph structure as used in the first dimension.
4. Repeat steps 2 and 3 until all redundancy symbols in all dimensions have been calculated.

The usage of multiple dimensions introduces a significant number of additional redundancy symbols. Therefore, in order to maintain the code rate, the number of redundancy symbols in each dimension has to be reduced, i.e. the ratio of redundancy symbols to source symbols per dimension, β, has to be reduced. If the number of dimensions is denoted as γ and the reduced ratio of redundancy symbols to source symbols for the multidimensional code is denoted as ϕ, then the code rate for multidimensional tornado codes can be calculated as

$$R = \frac{K}{N} = \frac{K}{K + \gamma \sum_{i=1}^{m+1} \phi^i K} = \frac{1}{1 + \gamma \sum_{i=1}^{m+1} \phi^i} \qquad (2)$$

Assuming an infinite number of bipartite graphs, i.e. m goes to infinity, equation (2) simplifies to

$$R = 1 - \frac{\gamma \phi}{1 + (\gamma - 1)\phi} \qquad (3)$$

Using the same argument as in section 2, the multidimensional Tornado code can recover from a random loss of $\gamma\phi/(1+(\gamma-1)\phi)$ fraction of its nodes with high probability. By choosing the number of dimensions to one, equation (3) obtains the form $R=1-\phi$, which is identical to equation (1).

The benefit of using multiple dimensions becomes clear when the decoding algorithm presented in section 2 is examined. The decoding algorithm states that the decoder requires a check node and all but one of its message nodes in order to recover a missing node. This means that if there are missing nodes to be recovered but no check node fulfills this criterion, the decoding algorithm will fail. When multiple dimensions are used, this problem may be overcome when the missing nodes are reconstructed in another dimension. The decoding may be possible in another dimension since the message nodes have been reordered and therefore depend on different check nodes. In other words, by using multiple dimensions the code is transformed into an iterative code, which iterates through the dimensions until no more nodes can be recovered.

5. Simulations and Test Results

A multidimensional Tornado code that used a symbol length equal to the IP packet payload was implemented. In this particular case, the symbol length was fixed to 1436 bytes. The packet size is, in the context of implementation and testing, not essential but represents a possible setup for delivering objects over a multicast network. The implementation, however, requires all IP packets to be of equal length. The code was tested with the number of dimensions ranging from one to six dimensions, the one-dimensional code being a standard Tornado code. The encoded IP packets were given to a DVB-H channel simulator, which introduced packet losses into the transmitted data. The simulator assumed that the IP packets were either entirely corrected by the lower layers in the network topology or entirely corrupted or lost. Therefore, the results presented below can be viewed as worst case scenarios, as improved implementations of the DVB-H subsystem could deliver partly corrected IP packets. The fractured data was then given to the decoder, which tried to recover from the losses.

The tests were performed with code lengths ranging from 400 to 4000, where a constant code rate of approximately 0.75 was used. The codes were created with only one bipartite graph, except for the standard Tornado code, since prior test results showed that using multiple dimensions had a greater impact on the error correcting performance than several layers of nodes had. Depending on the channel simulator settings, four different IP packet error rates (IP PER) were obtained in the uncorrected file. These IP PERs were approximately 9%, 10.5%, 13.5% and 15%. The simulations were run on a Pentium 4, 2.0 GHz processor, using a 30.36 MB mpeg file for testing. The implementation of the code was made in ANSI C on a UNIX platform.

In the figures showing the error correcting performances below, some of the lines are not continuous. The points where the lines break correspond to simulations in which the corresponding code was able to reconstruct all missing IP packets, thereby achieving an IP PER of 0%. As the figures are logarithmic, these points were not plotted.

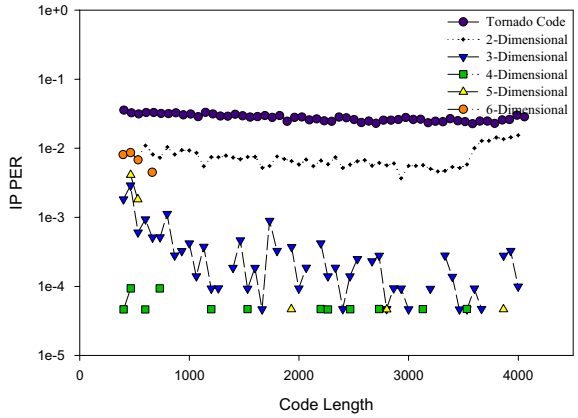


Fig. 2. Error correcting performances for multidimensional Tornado codes for an uncorrected IP PER of 9% on a simulated erasure channel.

Fig. 2 shows the IP PERs in the decoded files for the simulations in which the IP PER before correction was 9%. The figure demonstrates that the standard Tornado code was able to reduce the IP PER to approximately 2%, showing little improvement with the growing code length. The two-dimensional code performed slightly better, achieving decoded IP PERs of approximately 0.4%. The three-dimensional code dropped the IP PERs to below 0.01%, seldom correcting all symbols, however. The four-, five- and six-dimensional codes proved to be outstanding, with regard to the error-correcting performance, managing to reproduce the original file in almost all of the simulations.

The channel setting, which produced an IP PER of approximately 10.5%, was then investigated. The test

results of these simulations are illustrated in Fig. 3. The results show that the four-, five- and six-dimensional codes were unaffected by the increase in the received IP PER, while the rest of the codes experienced a small reduction in their error correcting performance.

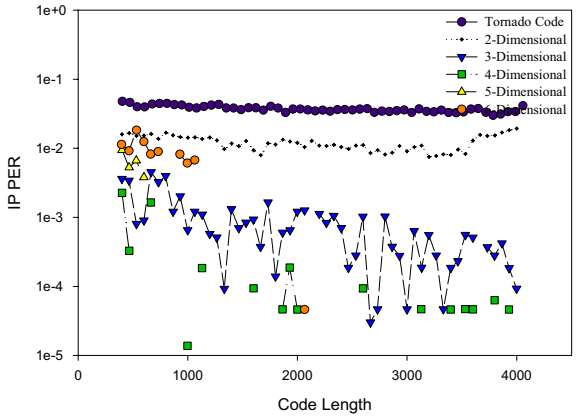


Fig. 3. Error correcting performances for multidimensional Tornado codes for an uncorrected IP PER of 10.5% on a simulated erasure channel.

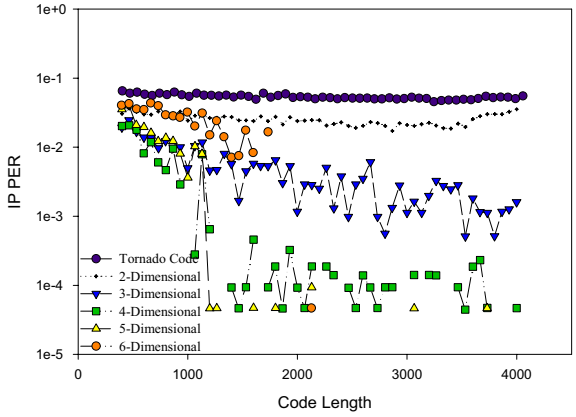


Fig. 4. Error correcting performances for multidimensional Tornado codes for an uncorrected IP PER of 13.5% on a simulated erasure channel.

The third channel setting produced an IP PER of approximately 13.5%. Fig. 4 shows that for a received IP PER of this magnitude, only the five- and six-dimensional Tornado codes were able to produce error free decoded files, and then only for slightly longer code lengths.

Fig 5. shows the test results for the codes when the IP PER before correction was approximately 15%. The figure reveals that the differences in the error correcting performances of the codes were small with the shorter code lengths. However, when the code length grew beyond 1700 encoding symbols, the five-dimensional Tornado code excelled at the error correcting performance, compared to the other codes.

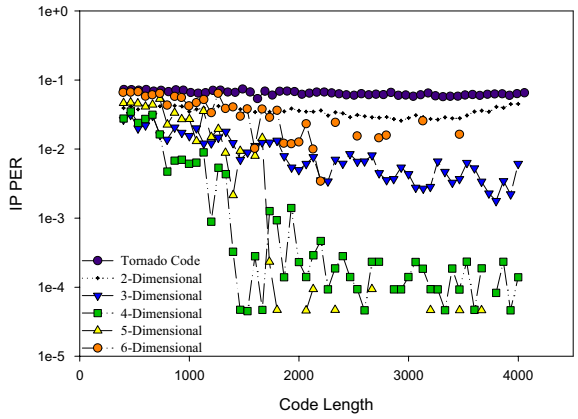
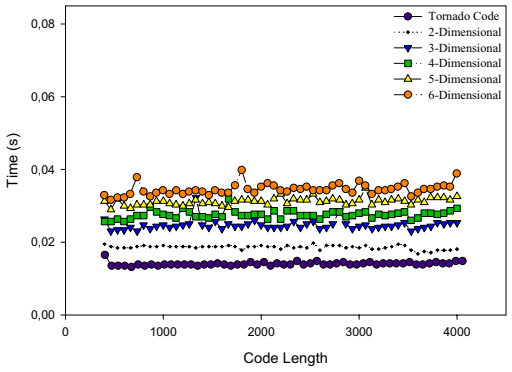


Fig. 5. Error correcting performances for multidimensional Tornado codes for an uncorrected IP PER of 15% on a simulated erasure channel.

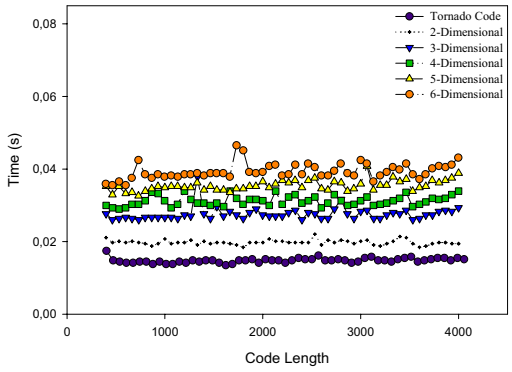
In each simulation, both the encoding and the decoding times were measured. The average encoding times per MB of source data for the different Tornado codes are listed in Table 1. As the table demonstrates, the encoding time grows linearly with the number of dimensions. During the simulations, it became clear that the encoding times are also dependent on the symbol length and on the code rate used for the codes.

TABLE 1
ENCODING TIMES PER MB OF SOURCE DATA

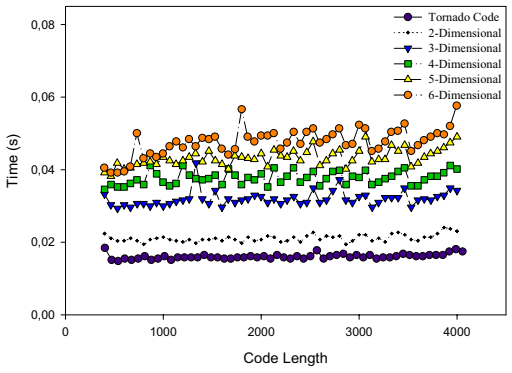
Code type	Time (ms)
Standard	29.974
2-Dimensional	38.208
3-Dimensional	50.725
4-Dimensional	64.888
5-Dimensional	77.734
6-Dimensional	87.615



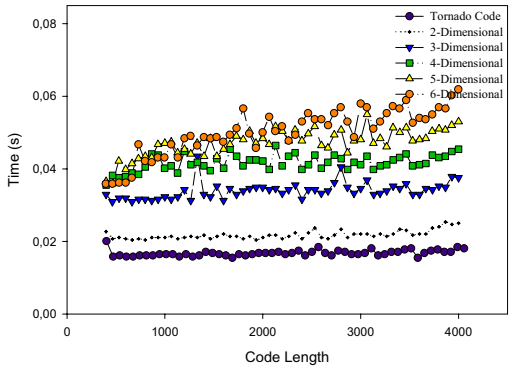
a) Decoding times for IP PER 9%.



b) Decoding times for IP PER 10.5%.



c) Decoding times for IP PER 13.5%.



d) Decoding times for IP PER 15%.

Fig. 6. Decoding times per MB of source data as functions of the code lengths for the simulations.

Fig. 6 illustrates the decoding times of the codes. Fig. 6 a) shows the decoding times for the simulations in which the uncorrected IP PER was 9%, Fig. 6 b) shows the decoding times for 10.5%, Fig. 6 c) shows the decoding times for 13.5% and Fig. 6 d) shows the decoding times for 15%. The graphs reveal that the decoding time is dependent on the number of dimensions and on the error rate in the received data. No other factors affect the decoding times.

6. Conclusions

In this paper, we showed with simulations that by designing the Tornado code structure with a Hyper code approach, the error correcting performance can be significantly improved. This is at the cost of slightly longer decoding times, since the decoder iterates through every dimension, correcting as many symbols as possible at a time before moving on to the next dimension. It should be noted, however, that this combination, or design methodology, does not create a new code, since this approach merely divides the Tornado code structure into several sections, or dimensions. What is interesting is that by using several dimensions, the error correcting performance is improved, compared to standard Tornado codes, which rely on several layers of nodes, rather than on several dimensions. In this paper, we have demonstrated that using multiple sets of redundancy symbols, where each set is calculated on the reordered message symbols, improves the probability of successful decoding. We have shown with simulations that this is true because the message symbols depend on different redundant symbols in each dimension and, therefore, if decoding is not possible in one dimension, the decoding may be possible in another dimension.

7. References

[1] J.W. Byers, M. G. Luby, M. Mitzenmacher, and A. Rege, "A Digital Fountain Approach to Reliable Distribution of Bulk Data", *Proceedings of ACM SIGCOM '98*, ACM Press New York, Vancouver, 1998, pp. 56-67.

[2] A. Hunt, S. Crozier, and D. Falconer, "Hyper-Codes: High-Performance Low-Complexity Error-Correcting Codes", *Proceedings of the 19th Biennial Symposium on Communications*, Kingston, Ontario, Canada, 1998, pp. 265-267.

[3] M.G. Luby, "LT Codes", *The 43rd Annual IEEE Symposium on Foundations of Computer Science (FOCS'02)*, IEEE Computer Society, Vancouver, 2002, pp. 271.

[4] M.G. Luby, M. Mitzenmacher, M.A. Shokrollahi, and D.A. Spielman, "Efficient Erasure Correction Codes", *IEEE Transactions on Information Theory*, IEEE Transactions on Information Theory Society, 2001, 47(2). pp. 589-584

[5] P. Elias, "Coding for two noisy channels", *Information Theory, 3rd London Symposium*, Butterworth's Scientific Publications, 1955, pp. 61-76.