Publication 5

Tuomas Tirronen and Jorma Virtamo. 2008. Finding fountain codes for real-time data by fixed point method. In: Proceedings of the 2008 International Symposium on Information Theory and its Applications (ISITA 2008). Auckland, New Zealand. 7-10 December 2008, pages 1244-1249.

International Symposium on Information Theory and its Applications, ISITA2008
Auckland, New Zealand, 7-10, December, 2008

Finding Fountain Codes for Real-Time Data by Fixed Point Method

Tuomas TIRRONEN and Jorma VIRTAMO

Helsinki University of Technology TKK
E-mail: {tuomas.tirronen,jorma.virtamo}@tkk.fi

Abstract

We study fountain coding-like transfer method over a packet erasure channel for data with real-time requirements, such as streaming video or voice. The proposed method uses a sliding window, which defines the current section of non-expired data. The coding is systematic, with new blocks entering the window being initially sent intact, followed by a possible correction packet. Two different strategies for correction packet sending are studied. In both methods correction packets are generated in fountain-coding like fashion using bitwise XOR-operation for addition of source blocks. The correction packets are constructed so as to maximize the probability of immediately decoding useful data at the receiver's end, based on the sender's belief on the receiver's state. After sending the correction packet this belief is updated. Fixed point iteration is used to find the stationary solution in long streams. The two methods differ in the details how the window is moved and how exactly the correction packets are generated. We study the performance of proposed methods and give theoretical and simulation results for both of the used approaches.

1. INTRODUCTION

Fountain codes and similar constructs provide an interesting and efficient alternative for traditional data transfer where the data are transferred in order and acknowledged. In fountain coding the order of the received data is not important and all of the sent symbols have statistically the same information value for the receiver. Thus the operation of true fountain codes depends only on the number of packets the receiver is able to collect. The code design specifies the amount of extra packets needed in addition to the original data size. First universal fountain codes were LT-codes [1]. Currently Raptor codes [2] are regarded as the state-of-the-art fountain codes.

For data which have real-time requirements, such as streaming video or audio, the number of outstanding undecoded blocks is limited. In traditional fountain codes the order of the data is not preserved thus they are impractical for coding real-time or streaming data. However, different methods have been employed to adapt fountain coding principle and existing fountain codes to real-time scenarios. Some of them also include a systematic part as in the proposed method, as for example in Raptor coding specification for the 3GPP project [3]. The same specification also defines Raptor codes for small message lengths between $k = 4$ and $k = 8192$, giving very good performance even for small message lengths.

In [4] the authors propose a method where a sliding window is used to mark the section of data where LT coding is employed and to overcome the problem of missing temporal ordering of the data. We propose two variants of a similar type of scheme, where we do not use LT coding but instead a systematic code with greedy correction packet generation in the sense presented in [5]. The proposed methods are fountain coding-like packet erasure coding methods, where packet generation itself is done by picking original data blocks and using modulo-2 addition, i.e., XOR operation, to generate a correction packet. Decoding is an iterative process similar to LT decoding, where blocks included in a received packet are compared to the set of already decoded blocks and possible new blocks are iteratively revealed by subtractions when possible. In comparison, efficient Raptor codes for small k use ML decoding [3], which equates to Gaussian elimination on erasure channels and is computationally more complex than iterative decoding.

We consider a scenario where data with real-time requirements, for instance streaming audio/video or VoIP, are to be transmitted over a packet erasure channel. We assume the data rate and delay requirements are defined by a higher level application, and these properties determine the size w of the sliding window. At a specific moment, the sliding window contains all of the currently useful, non-expired data. A given window is processed by sending new blocks intact after

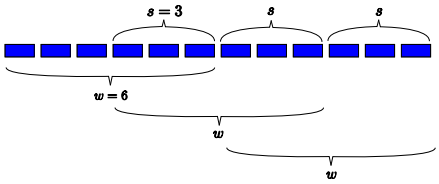

Figure 1: Sliding window depicted. The window determines the section of stream the encoder is working on. Window size parameter w is assumed to be given by the application.

which a correction packet is sent. Then the window advances s steps forward, new blocks are introduced into the other end of the window while equal amount of old blocks drop from the other end.

In our proposed erasure coding methods the correction packets are sent either i) deterministically after every s packets or ii) after each sent original block with a fixed probability P. We choose to construct the correction packets in these two cases in different ways, the respective optimization goals are: i) the number of blocks to be included in the correction packet from each sequence of s blocks and ii) the probabilities p_i used to include the ith packet from the window. The sender optimizes the packets given his belief on the state of the receiver. In particular, the correction packets are composed in a way which gives the highest probability for the decoder immediately to decode one missing block when a correction packet is successfully received. This optimization method is a greedy one, which means that the produced packets are only locally optimal, i.e., there is a possibility that the global optimum can be achieved with some other strategy.

After each correction packet is sent, the sender updates his belief on the state of the receiver, by updating the probabilities for each of the blocks inside the window to remain undecoded. The cycle of correction packet optimization, receiver state update and window move is repeated until stationarity by fixed point iteration. The resulting correction packet design is what we propose to be used for long streams of real-time data.

The performance of the proposed correction packet designs is investigated by simulations under different channel loss probabilities, using the residual probability for a data block to remain undecoded as the performance criterion.

2. SLIDING WINDOW AND CODING

We consider an infinite stream of equisized data blocks. A sliding window of w blocks encompasses all non-expired data and determines the section of the stream which the encoder works on, see Figure 1.

The encoder processes a window as follows: first each new block entering the window is sent as-is, constituting the systematic part of the code. After this, a correction packet is considered before the window is moved s steps (one step equals one block) forward from its previous position.

In general, a correction packet is generated by sampling file blocks and adding them together using XOR, i.e., the sent packet is a linear combination of blocks. The number of blocks included in a packet is called the packet *degree*. The two different approaches we study are:

1. Step size $s = w/2$ and a correction packet is always sent after the systematic part. The rate of the code is determined solely by w. The quantities to be optimized are the degrees d_1 and d_2, which give the number of blocks to be included from each of the half-windows. This scenario is depicted in Figure 1.

2. Step size $s = 1$, thus the window always advances one step at a time and after sending the new packet intact, a correction packet is sent with probability P. The used probability P determines the code rate. We optimize the probabilities p_i used to include the ith packet from the window.

Decoding is the same for both of the methods. A received block that was sent as-is requires no decoding at all. A received correction packet starts an iterative process, where all the blocks already decoded are first subtracted from the collected packet, possibly revealing a yet undecoded block. A correction packet which consists entirely of blocks that have already been decoded can be immediately discarded. If after the subtraction, which is also the XOR-operation, the packet still contains more than one block, it is saved in a buffer and used later if some of the included blocks become decoded and can be subtracted from it. Buffered packets which include blocks from the part of the stream which is no longer inside the sliding window can be removed from the buffer. We should emphasize, however, that the not-immediately-decodable packets are taken into account only in the final simulations, not in the analysis leading to the proposed correction packet designs as this would be too complicated.

The used channel model is packet erasure channel with independent packet losses. The channel erasure probability p is assumed to be known providing the sender sufficient information to update his belief on

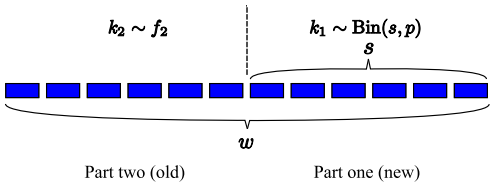

Figure 2: Illustration of the half-window step method. k_1 and k_2 are the number of undecoded blocks in the respective halves of the window.

the state of the receiver and to be able to optimize the correction packet composition. However, this assumption means that the designed codes are not rateless but designed for this specific channel, thus violating the original principle of fountain coding. The assumed independence of the packet losses is not realistic in most real-life scenarios but significantly simplifies the analysis and derivation of different coding methods.

The greedy optimization goal we use when optimizing the packet construction equates to finding such a packet where exactly one of the included blocks is not yet decoded by the receiver while all the other blocks are. Such a packet allows the receiver to calculate the original block $m_i = M \oplus s_{j_1} \oplus s_{j_2} \oplus \dots s_{j_k}$, where M is the received packet, and s_{j_i} correspond to the set of already decoded blocks also included in M.

Finally, the cycle of correction packet optimization, receiver state update and window movement is iterated until a stationary solution is found. This corresponds to a fixed point iteration and results in a correction packet design optimal in a greedy sense subject to some simplifications made to keep the analysis tractable.

In the next Section we present and analyze the half-window method, and in Section 4 we study the probabilistic sending strategy.

3. HALF-WINDOW STEP SIZE

When the step size $s = w/2$ we have the situation depicted in Figure 2. The window movement is followed by sending all the new s data blocks intact, i.e., the blocks in part one in Figure 2. Thereafter one correction packet is sent deterministically. The half-window movement results in two parts, where in the first half the missing number of new blocks, due to packet loss, is binomial and in the second half, the number of missing "old" blocks is something that we calculate using the fixed point iteration. This information can then be used to determine an erasure correction strategy.

The step size determines the true code rate. As after sending s new blocks one redundant packet is sent the rate is

$$R = \frac{s}{s+1} = \frac{w}{w+2}. \tag{1}$$

3.1. Greedy Optimization

The correction packets are formed by sampling d_1 and d_2 blocks uniformly at random from the respective halves of the window, i.e., we use constant packet degrees d_1 and d_2. In this method we directly seek the optimal d_1 and d_2, in contrast to the probabilistic method presented in Section 4.

Given the channel loss probability p, the number of blocks k_1 that are initially missing from part one is $k_1 \sim \text{Bin}(s,p)$. The corresponding distribution function is denoted $f_1(k_1)$. Further, let us denote the yet unknown distribution function of the number of missing packets k_2 for part two by $f_2(k_2)$. Note that in the half-window strategy k_1 and k_2 are independent which simplifies the analysis.

The probability that i undecoded blocks are picked into a degree d packet from a part of size s is (cf.[5])

$$P_i(d,k) = \frac{\binom{k}{i}\binom{s-k}{d-i}}{\binom{s}{d}}, \tag{2}$$

where k is the number of missing blocks ($s-k$ are already decoded). The explanation is combinatorial: from k missing blocks i are first chosen, followed by choosing the rest $d-i$ blocks from the $s-k$ already received ones, giving total number of ways to generate packet with i yet undecoded blocks. Finally this result is divided by the number of all possible packets.

The two cases we are interested in are those where exactly one yet undecoded block is picked either from the first or second sub-window and none from the opposite one. Thus the probabilities we need are of the type $P_0(d,k)$ and $P_1(d,k)$. According to our greedy optimization target our task is to find the degrees d_1 and d_2 that maximize the probability for generating a correction packet which the decoder can immediately decode to yield a new block. That is,

$$(d_1^*, d_2^*) = \arg\max_{(d_1,d_2)} Q(d_1,d_2), \tag{3}$$

where

$$Q(d_1,d_2) = \sum_{k_1=0}^{s}\sum_{k_2=0}^{s} \left(P_0(d_1,k_1)P_1(d_2,k_2) + P_1(d_1,k_1)P_0(d_2,k_2)\right) f_2(k_2) f_1(k_1). \tag{4}$$

is the probability for a correction packet to be such that exactly one yet undecoded block is picked from one of the half-windows.

The sender can update his belief on the unknown distribution f_2. Note that at the end of a full cycle, f_2 equals what f_1 was immediately after sending the correction packet, and f_1 changes only if the correction packet included exactly one undecoded block from the first half-window, and none from the second.

The probability for picking one yet undecoded packet from part one and none from part two is $P_1(d_1, k_1)P_0(d_2, k_2)$ as the considered events are independent. The conditional probability then becomes

$$U(k_1) = (1-p)Q(d_1^*, d_2^*) \sum_{k_2=0}^{s} f_2(k_2) \frac{P_0(d_2^*, k_2)P_1(d_1^*, k_1)}{Q(d_1^*, d_2^*)}$$

$$= (1-p) \sum_{k_2=0}^{s} [f_2(k_2)P_0(d_2^*, k_2)] \cdot P_1(d_1^*, k_1). \quad (5)$$

For the stationary state in a long stream we iterate the distribution function updating rule

$$f_2(k_1) = U(k_1) \cdot f_1(k_1 + 1) + (1 - U(k_1)) \cdot f_1(k_1),$$

where, as shown by (5), U depends functionally on f_2. Thus we have a relation $f_2 = F(f_2)$, a fixed point equation, which can be solved using iteration. When f_2 is solved we can calculate by (3) the optimal degrees, in the greedy sense, defining a proper erasure correction strategy.

3.2. Simulation Results and Discussion

The performance of the method was evaluated for different system parameters w and p. For each set of parameters, the optimal d_1^* and d_2^* were determined in advance as described above.

We simulated one stream of 200000 packets with different values of p. The simulation results are depicted in Figure 3. The used window sizes correspond to rates $0.75, 0.8, 0.83, 0.875$ and 0.91, respectively, from bottom to top. The performance naturally depends on the window size w, defining the code rate by (1). For example for $p = 0.10$ the best performance is obtained using $w = 6$, where the residual probability for a block to remain undecoded is 0.02, an improvement of 80% whereas with $w = 20$ the residual probability is 0.063 giving only 37% of improvement.

The degrees used in the correction packet packet generation depend on the channel loss probability. For low p in part 1 the used degree is half of w and for part 2 $w/4$, rounded up. As the loss probability rises, the used degrees become lower, ultimately even zero for part 2. The speed of convergence to zero is higher with larger window sizes. The described dependence of the degree on the channel loss probability is natural, as for a higher p there are more missing blocks after the systematic part, therefore there is a need to send packet with a lower degree in order to maximize the probability that there is only one undecoded block included in the correction packet.

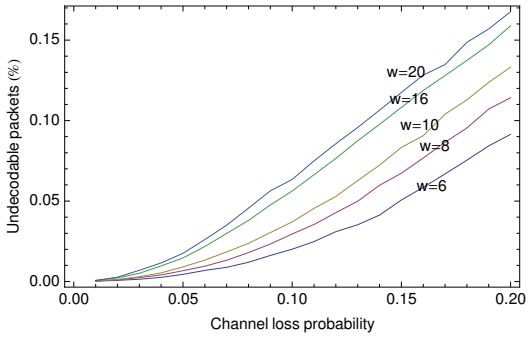

Figure 3: Residual error probability as a function of the channel loss probability. Different window sizes were used, from bottom to top $w = 6, 8, 10, 16$ and 20.

Direct comparison with other similar schemes, such as Raptor coding or the sliding-window LT coding presented in [4] is not made as the results present in the literature are not comparable to the presented simulations. However, it is expected that for example Raptor coding gives very good performance results if used as specified in [3]. Implementation and comparison to other schemes is left for later research.

4. PROBABILISTIC CORRECTION PACKETS

In our proposed second method we adopt a slightly different strategy for generating the correction packets. The correction packets are sent probabilistically after every original block with probability P. This is a chosen parameter defining the rate of the code

$$r = \frac{1}{1+P}. \quad (6)$$

Also, the way the correction packets are constructed is different. Instead of determining a fixed degree we now include each packet i in the correction packet with probability p_i. Our task is to optimize p_i and to this end we need to consider the probabilities of corresponding blocks to be undecoded, denoted by q_i. For older blocks in a window, the q_i are smaller, as possible corrections have already been made. The probabilities q_i grow towards the end of new blocks with $q_1 = p$.

4.1. Greedy Optimization

The probabilities p_i, or the vector of probabilities $\boldsymbol{p}$, define the probability to include a specific block in the correction packet. The optimization goal is to find $\boldsymbol{p}$ such that the generated correction packets are optimal in the greedy sense, where the packet must be composed of exactly one such block i which is undecoded and the other k included blocks are already decoded.

Using probabilities p_i and q_i, we can write the conditional probability for a particular block to be included in a correction packet and still be undecoded as $\mathcal{P}(\text{block } i \text{ is undecoded and included in } M) = p_i q_i$. The complement, $1 - p_i q_i$, defines the probability that either or both of these qualifications are not true. For the tractability of the analysis we make the assumption that the events of two different blocks i and j being undecoded are independent. Under this assumption we can write the probability for a new revealed block, given that a correction packet is sent and not lost in the channel, as

$$\begin{aligned} F = (1-p)\,&((p_1 q_1 \cdot (1-p_2 q_2)\cdots(1-p_w q_w) + \\ &(1-p_1 q_1)\cdot p_2 q_2 \cdot (1-p_3 q_3)\cdots(1-p_w q_w) + \\ &\vdots \\ &(1-p_1 q_1)(1-p_2 q_2)\cdots p_w q_w)) \\ &= (1-p)\cdot \sum_{i=1}^{w} \left(p_i q_i \prod_{j\neq i} (1-p_i q_i) \right). \end{aligned} \quad (7)$$

The probability vector that is optimal in the greedy sense is defined by

$$p_i^* = \arg\max_{p_i} \sum_{i=1}^{w} \left(p_i q_i \prod_{j\neq i} (1-p_i q_i) \right). \quad (8)$$

The solution is given by the following theorem:

Theorem 1. *The maximum of* (8) *is obtained by setting* $p_i = 1$ *for* $i = 1, \ldots, k$ *and* $p_i = 0$ *for* $i > k$*, where* k *is the smallest integer such that*

$$\sum_{i=1}^{k} \frac{q_i}{1-q_i} \geq 1.$$

Proof. (Outline) Equation (7) can be written in the form

$$\begin{aligned} F = \sum_{i\neq l} p_i q_i \prod_{j\neq i} (1-p_i q_i) + \left(1 - \sum_{i\neq l} \frac{p_i q_i}{1-p_i q_i} \right) \cdot \\ p_l q_l \prod_{j\neq l} (1-p_j q_j), \end{aligned} \quad (9)$$

for an arbitrary index l. This means that the function can be increased by setting p_l either 0 or 1, depending on whether the summand $\sum_{i\neq l} p_i q_i/(1-p_i q_i)$ is less or greater than 1. Further, since we have $q_1 \geq q_2 \geq \cdots \geq q_w$, we set as many p_is from the beginning to 1 as possible until the condition

$$\sum_{i=1}^{k} \frac{q_i}{1-q_i} < 1$$

is not true anymore. □

The sender again updates his belief on the state of the receiver, updating the probabilities q_i for each of the blocks inside the window and iterates, arriving at a fixed point solution. We start from some proper distribution for q_i, e.g. by setting all components initially to one and update these in a related fashion as in the half-window method. The update rule for q_i is

$$\begin{cases} q_1 \leftarrow p. \\ q_i \leftarrow \left(1 - (1-p)\cdot P \cdot p_i \prod_{j\neq k} (1-p_j q_j)\right) q_{i-1}, \end{cases}$$

After each update, a new set of the optimal p_i is calculated and the iteration is continued. Ultimately the iteration converges to a fixed point solution for the probabilities q_i. The corresponding set of probabilities p_i is then finally calculated by Theorem 1.

We use the last element q_w as the performance measure of this scheme. It defines the probability that a block is still undecoded when exiting the window, i.e., the residual probability of decoding failure.

4.2. Results and Discussion

In Figure 4 we have plotted a comparison between simulation and theoretical results for two window sizes, $w = 10$ and $w = 50$ with $P = 0.2$, corresponding to code rate of 0.83.

Again, for each set of parameters, the optimal composition of the correction packet, i.e., the optimal number of newest blocks to be included in it, was determined in advance.

Higher window sizes give slightly better performance. With $w = 10$ the solution using Theorem 1 gives a $\boldsymbol{p}$ vector of all ones, whereas with larger window sizes the picking probabilities of older blocks start to switch to 0 as the loss probability increases. Naturally, if the window is large enough to contain $p_i = 1$ up to some k and $p_i = 0$ for $i > k$, the window size can be decreased to $w = k$ without loss of performance.

A comparison with the results in Section 3.2 suggests that the half-window method and picking fixed degrees gives somewhat better results. For example when $p = 0.1$ the half-window method gives residual

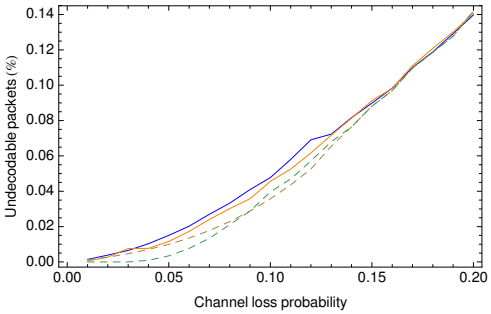

Figure 4: Residual error probabilities when $P = 0.2$. The dotted lines are theoretical results, two different window sizes were used $w = 10$ (upper lines) and 50.

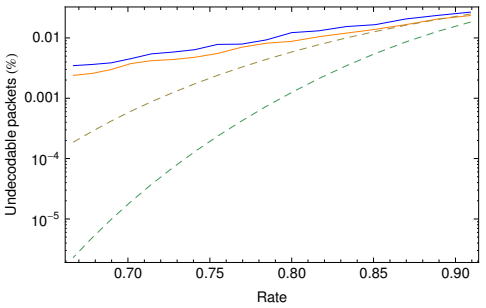

Figure 5: Theoretical and simulated dependence of the residual error on the code rate, channel loss probability $p = 0.05$.

probability of 0.037 while the probabilistic model, with large enough window, only 0.045.

5. CONCLUSIONS

We proposed and analyzed two methods for fountain coding-like erasure correction for real-time data. Both of the methods include solving a fixed point equation for determining the optimal strategies for correction packet generation. The half-window method has the advantage that the calculation is easy using the independence of distributions. Relaxing the limitation $s = w/2$ is possible but needs additional work. The second method has the possibility for altering the rate parameter of the code continuously. However, the probabilistic emission of correction packets can result in some randomness in the distribution of correction packets over the stream, whereas the half-window method does not have this problem. The results suggest that the fixed degree-method is more efficient.

When considering the optimality of the presented methods a few issues should be considered. First of all, the objective (7) for the probabilistic method is an approximation since some interdependencies have deliberately been omitted for the sake of tractability. Further, the optimization method used for both schemes is a greedy one, which considers only the immediate benefits of a received packets. Due to the iterative nature of the decoding process, there could be some better choice for the correction packet generation. The last point is the that the chosen methods for correction packet generation, i.e., by either picking the packets randomly from the respective halves of the window with given degrees or picking the packets independently from the whole window with given site-dependent probabilities are not necessarily optimal.

A task for further research is also the extension of the half-window method to the case where the window is divided into more parts as well as studying the probabilistic dependencies between these parts.

ACKNOWLEDGEMENTS

The authors would like to thank Dr. Aleksi Penttinen for useful discussions and for providing the proof for Theorem 1. The work of Tuomas Tirronen was financially supported by GETA graduate school and by the project ABI funded by Tekes.

References

[1] M. Luby, "LT codes," in *The 43rd Annual IEEE Symposium on Foundations of Computer Science*, 2002, pp. 271–282.

[2] A. Shokrollahi, "Raptor codes," *IEEE Transactions on Information Theory*, vol. 52, no. 6, pp. 2551–2567, 2006.

[3] "3GPP TS 26.346 v7.6.0, 3rd Generation Partnership Project, Technical Specification Group Services and System Aspects, Multimedia Broadcast/Multicast Service (MBMS), Protocols and codecs," Tech. Rep., Dec. 2007.

[4] M. C. Bogino, P. Cataldi, M. Grangetto, E. Magli, and G. Olmo, "Sliding-window digital fountain codes for streaming of multimedia contents," in *IEEE International Symposium on Circuits and Systems*, May 2007.

[5] T. Tirronen and J. Virtamo, "Greedy approach for efficient packet erasure coding," in *Proceedings of 5th International Symposium on Turbo Codes and Related Topics*, Lausanne, Switzerland, Sep. 2008, pp. 344–349.