Raptor Codes for P2P Streaming

Philipp M. Eittenberger
Faculty of Information Systems
and Applied Computer Science,
Otto-Friedrich University,
Bamberg, Germany

Todor Mladenov
Department of Information
and Communications,
Gwangju Institute of Science and Technology,
Gwangju, South Korea

Udo R. Krieger
Faculty of Information Systems
and Applied Computer Science,
Otto-Friedrich University,
Bamberg, Germany

***Abstract*—In this paper, we present a first analysis of the application of Raptor codes in the domain of P2P streaming. With the help of fountain codes, such as Raptor codes, it is possible to completely omit content reconciliation in P2P networks. Hereby, the scheduling complexity of the data dissemination is greatly reduced. The contributions of the paper are the following: First, we present our implementation of the Raptor code used in the performed experiments and elaborate the application of the Raptor code in the scenario of P2P streaming. Second, we investigate the choice of the prevalent parameters, necessary to achieve the best trade-off between performance, computational complexity and resilience of the Raptor code. We use the obtained results to evaluate the general feasibility of using Raptor codes to improve the performance of P2P streaming networks. In addition, we report some insights arising from the practical experience with Raptor codes.**

I. Introduction

Peer-to-Peer (P2P) streaming applications have attracted a lot of attention in recent years. Numerous scientific studies investigate their properties, large research projects have been founded to develop prototypes (e.g. NapaWine[5] or PPNext[6]), but more important, real systems have also been deployed successfully. Today, the most popular representatives among those systems are PPLive[7], PPStream[8] and SopCast[9]. These P2P streaming applications are able to serve simultaneously up to hundreds of thousands of users nowadays. Two different service types can be distinguished within P2P streaming: A *video on demand (VoD)* system provides users with VCR functionality, e.g. stop, rewind or fast forward of the video. In contrast, by *live streaming* the users have a more TV-like experience, where all users view the same playback time within a certain range of delay. Regarding the system architecture and in particular the implementation of the data dissemination, the systems can be coarsely divided into two main groups: *Mesh-pull* systems build an unstructured overlay, hence "mesh", and each peer requests, "pulls", the data from other peers. *Tree-push* systems explicitly construct a dissemination overlay and "push" the data along the constructed "trees". In reality, many "hybrid" combinations of both approaches exist, but all of the successfully deployed systems construct a dense mesh of neighboring peers; since due to the high fluctuation rate of the participants in a P2P network the organization and maintenance of one or several dedicated dissemination tree(s) have been proven to be too computationally costly and vulnerable[15].

Regardless of the type of P2P system, two optimization problems have to be solved to achieve high data throughput in the network: At first, the choice which pieces, also called *chunks*, should be requested from or send to other peers. Then subsequently, the selection which peers should be chosen for the data transfer. The optimal solution of both problems is inherently NP-complete. Therefore, the problems are computationally intractable, i.e. for the time being, there is no solution with polynomial run time. Due to that reason, much research has been done to develop algorithms that yield good approximations. But even if the algorithm yields good estimates, a further requirement must be considered in addition: The algorithm must be applied in a distributed manner, as centralized approaches do not scale for large, distributed and highly dynamic systems like P2P networks.

Various dissemination algorithms for P2P streaming have been proposed that address these two problems (see [10] for further references). However, by using one particular class of forward error correction (FEC) codes, it is possible to omit one of the two optimization problems. Fountain codes could reduce the problem to a bandwidth allocation problem for which well performing distributed approximations are available (e.g. [22]). Due to their desirable property of *ratelessness*, they have the ability to generate theoretically an unlimited amount of uniquely encoded data on-the-fly. This property eliminates completely the need for content reconciliation, as no redundant content exists in the network. Chunk scheduling would not be needed any more, since a receiving peer could restore the original data needing just a slightly larger amount of encoded data from any set of peers. Another benefit is the improved exploitation of low quality neighbors, i.e. peers with a slow bandwidth or a high delay connection. Due to the real-time constraint, common P2P streaming networks require highly sophisticated scheduling strategies in order to take advantage of the upload capacity of such peers. In contrast to Fountain code enabled P2P streaming applications, where every received symbol of a block is useful for the decoding of the original data. Thereby, it is easier to make use of the resources of low quality peers. To summarize, fountain codes

The final version of this paper appeared in the 20th Euromicro International Conference on Parallel, Distributed and Network-Based Processing (PDP), 2012. DOI: 10.1109/PDP.2012.80

are the ideal supplement for an unreliable but lightweight transport protocol like UDP to increase the dissemination performance of P2P networks.

However, realizing this promise of erasure coding is not a straightforward task: For instance, there are patent concerns regarding the use of fountain codes that scare off potential users. Despite the fact that one algorithm for a Raptor code, a potential candidate of the class of fountain codes, is described e.g. in RFC 5053 [14] and also the 3GPP standard MBMS (Multimedia Broadcast/Multicast Services) [1] includes a systematic Raptor code in their specifications for content delivery, there is nowhere to find a free implementation of a Raptor code. Perhaps because the underlying theory cannot be understood so easily. We will present our insights and experiments regarding the use of Raptor codes in the domain of P2P streaming. The paper is organized as follows: We introduce Raptor codes in Section II. In addition, we present the use case of a Raptor code implementation in combination with P2P streaming. The following experiments in Section III are conducted to obtain the fundamental parameters needed for future deployment. Subsequently, we discuss related work in Section IV. Finally, we conclude the paper in Section V.

II. Raptor Codes

As mentioned before, fountain codes have the ability to generate potentially unlimited, uniquely encoded data on-the-fly. **Rap**id **Tor**nado codes belong to the class of fountain codes. They have been developed by Amin Shokrollahi [19] as an advancement of Tornado codes [13]. Raptor codes represent an improvement over Luby transform codes (LT codes), which have been invented by Michael Luby [11] and represent the first practical class of a fountain code with near optimal error correction functionality. In this section we provide a brief description of the Raptor coding operations. For the theoretical details of Raptor codes, we would like to refer the reader to [19] and for further details on practical aspects to [12].

Given a data file divided into T data blocks each consisting of a set of k *input symbols* $\Omega = (x_1, ..., x_k)$ with symbol size l bytes, a fountain code can theoretically provide an unlimited supply of uniquely encoded *output symbols* $(z_1, ..., z_n)$. This desirable property is called *ratelessness*. In this context the term "symbol" represents just a data unit. The decoder can recover the source symbols from any set of $\Theta = k(1 + \varepsilon)$ encoded symbols, with Θ slightly larger than k. The surplus of symbols $\varepsilon \times k$ is called the overhead of the code and these additional symbols are also called *repair symbols*. Fountain codes do not guarantee the successful decoding by k symbols, however, the decoding failure probability decreases with each additional symbol. Another desirable aspect is the possibility that each receiver is able to decode the encoded symbols generated by different rateless code encoders, if both use the same rateless code and operate on the same input symbols. Due to that property, content reconciliation can be avoided, i.e. there is no need for the chunk scheduling in P2P networks. Our implementation of a Raptor code follows the specification given in [1]. The Raptor code was implemented in plain C code; therefore, it is quite fast and the timing accuracy needed for the measurements is reasonably high. The specification in [1] uses a combination of a low density parity check (LDPC) precode with an LT code. The rateless property of the Raptor code is a result of the LT code, while the increased performance is due to the LDPC code. The encoding consists of two phases: In the *pre-code phase* a matrix A is used to generate the *intermediate symbols* $(c_1, ..., c_\Theta)$ from the k source symbols. Matrix A consists mainly of three submatrices: a LDPC matrix, a higher density check matrix and a LT code matrix. In the second phase the LT encoder selects randomly a set of intermediate symbols $\phi = (c_1, ..., c_m)$ with $m < k$ and generates the output symbols $(z_1, ..., z_\Theta)$. For each encoded symbol, the encoder chooses randomly a degree d between 1 and k from a specifically designed degree distribution and selects also randomly a neighborhood of connected d intermediate symbols. Each output symbol z_x is then generated by XORing the set of chosen intermediate symbols. To yield the random values, a pseudo random number generator (PRNG) can be used. By relying on a pseudo-random process, some initial value has to be determined that serves as the seed for the PRNG. Regarding the decoding of the encoded symbols the receiver needs to know the chosen degree d and the set of chosen intermediate symbols. One possibility to inform the decoder is given by the transmission of the initial value that was used as a seed for the PRNG. However, one very important condition must be met to omit the content reconciliation completely. Each encoder, i.e. each peer, should use a unique seed value for its PRNG. Otherwise, encoders with the same seed value would produce the same output symbols. Such redundant encoded symbols would increase the overhead rate of the Raptor code.

Upon reception of a given threshold $\Theta = k(1 + \varepsilon)$ of encoded symbols, the receiver starts the decoding process. The threshold is the minimum amount of symbols that make the pre-code matrix A invertible and thereby, the decoding successful. At first, the decoder starts with the application of the pre-code on the set of received symbols and afterwards, it applies the LT code. To restore the intermediate symbols, the pre-code matrix A must be inverted:

$$\phi^T = e^T \times A^{-1} \tag{1}$$

where $(.)^T$ is the transpose of $(.)$ and e is the vector of encoded symbols. The matrix inversion is the computationally most expensive operation at the decoder, accounting for up to 92 % of the computing time. The source symbols are then obtained according to:

$$\Omega = G_{LT} \times \phi \tag{2}$$

with G_{LT} as generator matrix of the LT code.
In summary, the excellent performance, the minimal overhead, approaching very closely an ideal fountain code, and the ability to efficiently support dynamically adjusted block sizes, makes Raptor codes the ideal candidate in helping P2P networks to get rid off the chunk scheduling problem.

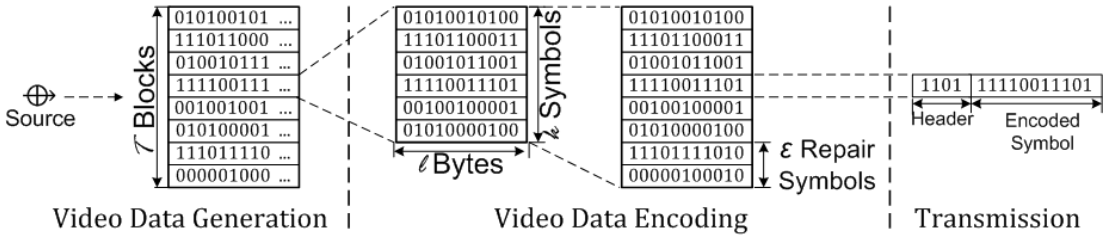

Figure 1. Encoding and packetization of the video data for P2P streaming

A. P2P Streaming with Raptor Codes

To enable the P2P video data dissemination with Raptor codes, the encoded symbols must be packetized and annotated with information needed for the decoding process. This procedure is coarsely illustrated in Figure 1: At first, the source data is segmented into T blocks. The size of each block k in terms of symbols and the size of each symbol l can be dynamically adjusted due to the rateless property of Raptor Codes. Every block is marked with a unique ID (BlockID). The Raptor code is then applied independently on each block. Every block consists of a number k of source symbols of size l bytes. If the last symbol of a block would not consist of l bytes, it is padded with 0s to l. All the symbols of one block get consecutive IDs (SymbolID), which indicate the seed values for the random number generator that was used to determine d. The header for each encoded symbol includes the BlockID, the SymbolID and the symbol size l. This information is necessary to ensure that the receiving peer can match the symbol to the correct block and is able to decode the data successfully. When the receiving peer reaches the threshold of Θ encoded symbols, it starts to decode the particular block. If the source block could be restored successfully, the peer will indicate the transmission stop for the particular block by sending ACK messages to all participating peers. If the decoding was not successful, the receiver waits for additional symbols and retries the decoding again. After the successful decoding, the peer encodes the data again with its own unique seed values for the further dissemination to other requesting peers. Before the encoding and transmission of one block can start, the participating peers must agree upon the block size and the symbol size. Therefore, a special message containing the block header information is exchanged. As long as the symbol size l is much bigger than the header, the additional information of the exchanged message introduces only a small overhead. The overhead rate Θ can be determined by the chosen block size. The goal of our work is to find the parameters that maximize the transmission speed and meanwhile decrease the decoding speed in order to yield an excellent video playback quality. Since we are especially interested in the performance that can be achieved in practice, we will investigate in the following experiments the best trade-off between performance, computational complexity and resilience of the Raptor code.

III. Measurement Results

In order to exploit the potential of Raptor codes for P2P streaming and hereby, reduce the scheduling complexity and increase the performance of the P2P data dissemination, a few important questions have to be answered first: How big should the symbol size and the block size be chosen? How many repair symbols are needed in each block? And does the number of peers, from which one peer receives concurrently data, play a role? Since we are mainly interested in the basic parameter settings and the achievable performance, we use a simple simulation setup to provide answers to these questions and thereby, build the foundation for the future application of Raptor codes in the domain of P2P streaming.

A. Symbol Size

At first, the influence of the symbol size l on the decoding time needs to be investigated. Theoretically, Raptor codes have a linear decoding time $O(k\ log(1/\varepsilon))$; therefore, by choosing a symbol size and block size as large as possible, the total amount of decoding time could be decreased in theory. However, as our implementation relies heavily on the matrix inversion algorithm, which accounts for up to 92 % of the computing time, the theoretical optimum can not be reached for large block sizes. In addition, as the communication between sender and receiver takes place in the Internet with the prevalence of Ethernet technology in the customer premises network, there is the artificial border of the maximum transmission unit (MTU). As most Ethernet LANs use a MTU of 1500 bytes, it would increase the complexity, if the symbol size is chosen larger than approx. 1450 bytes (= Ethernet MTU - IP header [=20 bytes] - TCP/UDP header [=20/8 bytes] - application header); because symbols with $l >$ (MTU - headers) bytes need to be fragmented at the sender and reassembled at the receiver.

Since we are especially interested in the influence of currently available commodity hardware on the encountered time lags, we performed the experiments on an Intel Pentium i7-860 CPU with 2.8 Ghz and Linux as operating system. As we do not know the underlying distribution, we used the bootstrap method to obtain very conservative confidence intervals for the mean values. The obtained samples $\overrightarrow{X} = (X_1, ..., X_n)$ are regarded as an approximation of the true, unknown distribution. Under the condition that the random variables are independent and identically distributed (iid), the Glivenko–Cantelli theorem claims that the empirical distribution function converges with

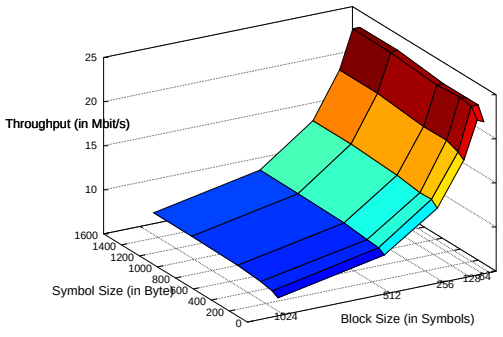

Figure 2. Encoding Throughput

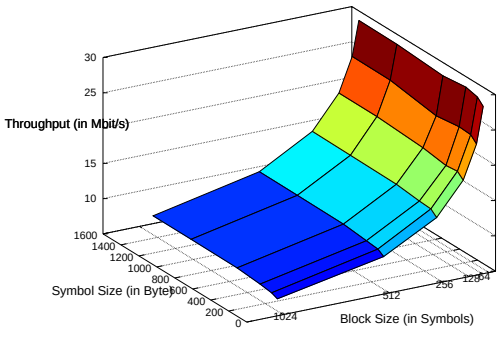

Figure 3. Decoding Throughput

probability 1 to the true CDF with a growing number of observations. However, the computed confidence intervals, even for a confidence level of 99 %, are too small to be visible in the plot. Our implementation reaches with such off-the-shelf hardware already encoding/decoding speeds of several Mbits per second (see Figure 2 and Figure 3): On the i7 machine, using only one CPU thread, the Raptor code reaches regardless of the symbol size l with a block size of $k = 32$ an encoding speed of almost 24 Mbits/s and a decoding speed of 29 Mbits/s. As one can observe, the influence of the symbol size on the encoding and decoding speed is minimal, yet, larger symbol sizes increase the speed slightly. At the moment, this performance would be already enough for current P2P streaming applications, which serve channels with playback rates of several hundreds of Kbits/s up to 2 Mbits/s. Since our implementation was not optimized for encoding/decoding speed, we are optimistic to yield even better results through optimizations, but we leave this task for future work.

B. Block Size

In conjunction with the symbol size, the optimal block size needs to be determined, in order to achieve a good trade-off between the overhead of repair symbols, the amount of time needed for the decoding and the delay the receiver encounters while waiting to reach the decoding threshold. The outcome of this investigation seeks to minimize the start up delay and the amount of time needed for the coding and decoding process. Hence, if the block size is chosen very large, the receiver will encounter an excessive start up delay. Smaller block sizes at the beginning of the transmission would decrease the startup delay, but how efficient are they in terms of decoding speed and overhead rate? In our experiments we investigated a range of block sizes with $k = 32, 64, 128, 256, 512, 1024$ and a symbol size $l = 64, 128, 256, 512, 1024, 1448$ bytes. For every combination of block and symbol size, randomly chosen video data was encoded and decoded on the test machine. The resulting throughput of the encoding and decoding process are depicted in Figure 2 respectively in Figure 3. The effect of the higher complexity of the matrix inversion with increasing block sizes is clearly visible. Hence, to boost the encoding/decoding speed, small block sizes are beneficial. However, block sizes chosen too small ($k \leq 32$) have an adverse effect on the encoding performance, as the content switching overhead for the CPU increases. In addition, one problem remains the overhead rate of repair symbols that are necessary to ensure the successful decoding. For block sizes with $k = 10$ symbols ε can be as high as 110 %, whereas ε drops to less then 1 % for block sizes greater than 1600 symbols. From the perspective of requiring a small overhead rate, large block sizes are necessary. However, if the block size is chosen too large, the receiver will have to wait for the reception of the Θ symbols, leading to longer decoding delays. The two types of P2P streaming, VoD and live streaming, require different approaches in regard to this choice. For VoD the block sizes should start small, but then, increase fast to a certain threshold to be able to exploit the benefits of a reduced overhead. In contrast, in the case of live streaming, block sizes chosen too large would imply a negative effect on the playback lag. In this setting the block size needs to be chosen quite small to avoid too large playback delays. But both cases require always a compromise between encoding/decoding speed and coding overhead.

C. Number of Repair Symbols

Obviously, the next open question concerns the number of repair symbols needed for the successful decoding of each block. Since each additional repair symbol increases the overhead, i.e. the threshold Θ that ensures a successful decoding with high probability, the right number has to be chosen. However, even if the decoding should fail, a retry with a slightly higher number of Θ will yield success with very high probability. We initially performed experiments with

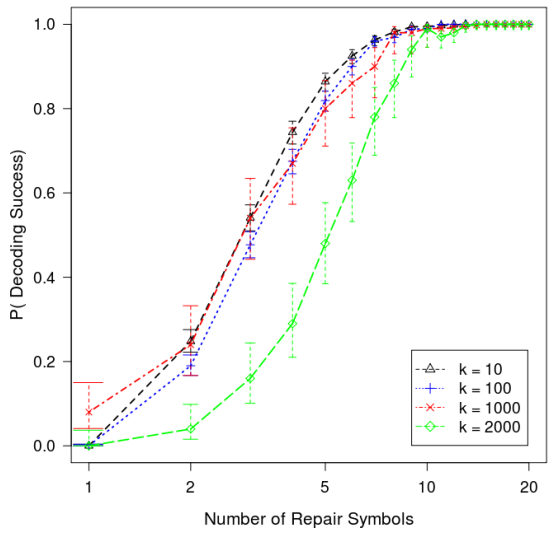

Figure 4. Decoding success depending on the number of repair symbols

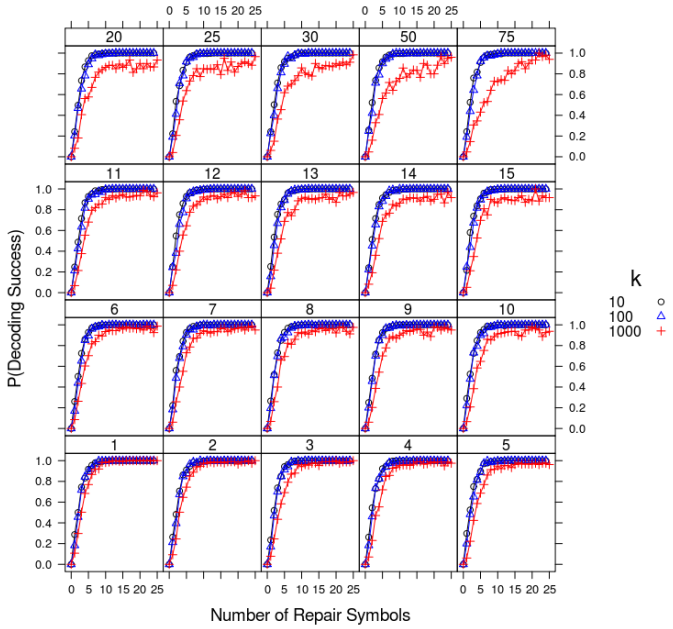

Figure 5. Decoding success depending on the number of repair symbols in the multi peer scenario. The number of sending peers is given in the panel above each graph.

$k = 10, 100, 1000$ and 2000 symbols and $l = 1448$ bytes. The results of the experiments are illustrated in Figure 4. We use very conservative confidence intervals based on Wilson's score interval[21] with a confidence level of 95 %. Gasiba *et al.* [3] report that on average $k + 2$ symbols are sufficient to recover from transmission errors. They investigate source block sizes of $k = 100, 1000$ with $l = 512$ bytes. As the the symbol size l should have no influence on the decoding probability, the results of our experiments do not confirm their Θ value. The minimum threshold of additional repair symbols that ensures the decoding success with a probability of 99.9 % is given in Table I. Nevertheless, the obtained overhead rate is still very close to that of an ideal fountain code for larger block sizes. Figure 6 depicts the outcome of further experiments regarding the necessary overhead rate for a decoding success of 99.9 %. In this experiment we investigated the combinations of block sizes with $k = 32, 64, 128, 256, 512, 1024$ and symbol sizes $l = 64, 128, 256, 512, 1024, 1448$ bytes. In accordance with the theory, it can be observed that the symbol size l has no influence on the decoding probability. The obtained overhead rate drops from 34.37 % for $k = 32$ to acceptable 1.56 % for $k = 1024$ (see also Table I).

D. Peer Neighborhood Size

As each peer receives data from a multitude of peers, the optimal number of concurrent data transmissions for one block is an important parameter. Theoretically, there should be no difference compared to the scenario of a single encoder as long as all peers use unique seed values for their PNRGs. But as depicted in Figure 5, we could observe an increase of needed repair symbols for the case of $k = 1000$ symbols growing with the number of peers. This would be a normal consequence, if there is no guarantee that all the different peers use unique seed values, because encoders using the same seed produce the same redundant output symbols. However, we are ensuring the uniqueness of the seed values in our controlled simulation environment. Since the degree distribution given in [1] was not designed for such a scenario, we assume that the distribution might not provide consistent performance over all possible draws. But the increase of ε is not worrying in terms of the overhead rate for large block sizes, rising for $k = 1000$ and the decoding probability of 99.9 % from 16 symbols, i.e. $\varepsilon = 1.6\%$, for a single sender to 30 symbols for 75 peers, and thus, $\varepsilon = 3\%$.

IV. Related Work

The usage of forward error correction codes in the domain of video transmission has been proposed more than 20 years ago [16]. However, highly efficient FEC that enable this scenario have been made available only in recent years. Raptor codes are already used for media broadcasting and unicast streaming (e.g. [2] or [17]). The first work proposing the usage of rateless codes to avoid the chunk scheduling problem in P2P streaming networks was presented by Wu and Li [22]. In this study they use LT codes in a simulation setup with a prototypical P2P streaming application to evaluate the general feasibility. Suh *et al.* [20] construct an analytical model for a VoD P2P system that incorporates also the use of rateless codes. Grangetto *et al.* [4] investigate the application of LT codes for P2P streaming albeit in a pure simulation study too. More recently, Oh *et al.* [18] present a P2P streaming prototype that uses Online codes and propose exploiting the rateless property of fountain codes to adapt the symbol size dynamically in order to avoid excessive start up delays.

Table I
MINIMUM AMOUNT OF REPAIR SYMBOLS NECESSARY TO ACHIEVE A DECODING SUCCESS OF 99.9 %

k	10	32	64	100	128	256	512	1000	1024	2000
# repair symbols	11	11	11	11	12	13	15	16	16	17
ε	110 %	34.37 %	17.18 %	11 %	9.37 %	5.07%	2.92 %	1.6 %	1.56 %	0.85 %

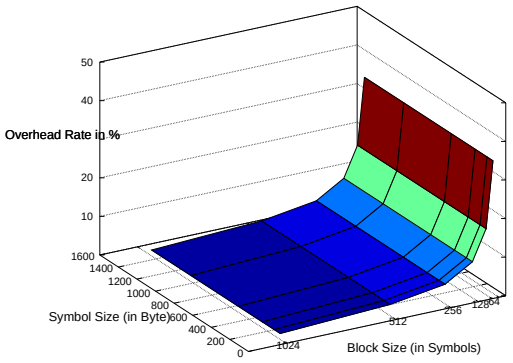

Figure 6. Overhead rate for a decoding success of 99.9 %

V. CONCLUSION

Compared to Online or LT codes, Raptor codes have a superior performance in terms of coding speed and overhead rate. Our first insights regarding the application of Raptor codes in the domain of P2P streaming and the results of our experiments are also promising; they require only a small overhead rate for large block sizes and are able to support the necessary streaming rates. To summarize, this paper introduces the application of Raptor codes for P2P streaming. In particular, we present a first analysis of the parameters needed for the successful future deployment and report the performance that can be expected by using Raptor codes. However, the obtained results are not only limited to P2P streaming networks, but are valid for all types of applications needing advanced forward error correction. We conclude that the general feasibility of this approach is given and that Raptor codes provide an excellent alternative to reduce the complexity of the data dissemination in P2P networks.

ACKNOWLEDGMENTS

The authors acknowledge the partial financial support by the project COST IC0703.

REFERENCES

[1] 3GPP TS 26.346, technical specification group services and system aspects; multimedia broadcast/multicast services (mbms); protocols and codecs. 2007.

[2] P. Cataldi, M. Grangetto, T. Tillo, E. Magli, and G. Olmo. Sliding-window raptor codes for efficient scalable wireless video broadcasting with unequal loss protection. *IEEE Transactions on Image Processing*, 19(6):1491 –1503, june 2010.

[3] T. Gasiba, T. Stockhammer, and W. Xu. Reliable and efficient download delivery with raptor codes. *6th International ITG-Conference on Source and Channel Coding (TURBOCODING)*, pages 1–6, 2006.

[4] M. Grangetto, R. Gaeta, and M. Sereno. Rateless codes network coding for simple and efficient p2p video streaming. In *Proceedings of the 2009 IEEE international conference on Multimedia and Expo*, ICME '09, pages 1500–1503, Piscataway, NJ, USA, 2009. IEEE Press.

[5] http://napa wine.eu. Napa Wine.

[6] http://www.p2p next.org/. P2P-Next.

[7] http://www.pplive.com. PPLive.

[8] http://www.ppstream.com. PPStream.

[9] http://www.sopcast.com. SopCast.

[10] C. Liang, Y. Guo, and Y. Liu. Investigating the scheduling sensitivity of p2p video streaming: An experimental study. *IEEE Transactions on Multimedia*, 11(3):348 –360, april 2009.

[11] M. Luby. LT codes. In *The 43rd Annual IEEE Symposium on Foundations of Computer Science*, pages 271–280. IEEE, 2002.

[12] M. Luby, T. Gasiba, T. Stockhammer, and M. Watson. Reliable multimedia download delivery in cellular broadcast networks. *IEEE Transactions on Broadcasting*, 53(1), 2007.

[13] M. Luby, M. Mitzenmacher, A. Shokrollahi, D. Spielman, and V. Stemann. Practical loss-resilient codes. In *Proceedings of the twenty-ninth annual ACM symposium on Theory of computing*, STOC '97, pages 150–159, New York, NY, USA, 1997. ACM.

[14] M. Luby, A. Shokrollahi, M. Watson, and T. Stockhammer. RFC 5053. 2007.

[15] N. Magharei, R. Rejaie, and Y. Guo. Mesh or multiple-tree: A comparative study of live p2p streaming approaches. In *26th IEEE International Conference on Computer Communications.*, INFOCOM '07, pages 1424–1432, 2007.

[16] A. J. McAuley. Reliable broadband communication using a burst erasure correcting code. In *Proceedings of the ACM symposium on Communications architectures & protocols*, SIGCOMM '90, pages 297–306, New York, NY, USA, 1990. ACM.

[17] T. Mladenov, S. Nooshabadi, and K. Kim. Mbms raptor codes design trade-offs for iptv. *IEEE Transactions on Consumer Electronics*, 56(3):1264 –1269, 2010.

[18] H. R. Oh, D. O. Wu, and H. Song. An effective mesh-pull-based p2p video streaming system using fountain codes with variable symbol sizes. *Computer Networks*, 55(12):2746–2759, 2011.

[19] A. Shokrollahi. Raptor codes. *IEEE Transactions on Information Theory*, 52:2551–2567, June 2006.

[20] K. Suh, C. Diot, J. Kurose, L. Massoulie, C. Neumann, D. Towsley, and M. Varvello. Push-to-Peer Video-on-Demand System: Design and Evaluation. *IEEE Journal on Selected Areas in Communications*, 25(9):1706–1716, Dec. 2007.

[21] E. B. Wilson. Probable inference, the law of succession, and statistical inference. *Journal of the American Statistical Association*, 22(158):209–212, 1927.

[22] C. Wu and B. Li. rStream: resilient peer-to-peer streaming with rateless codes. In *Proceedings of the 13th annual ACM international conference on Multimedia*, MULTIMEDIA '05, pages 307–310, New York, NY, USA, 2005. ACM.