Full text available at: http://dx.doi.org/10.1561/0100000060

Raptor Codes

Raptor Codes

Amin Shokrollahi

EPFL
Station 14
Lausanne 1015
Switzerland
amin.shokrollahi@epfl.ch

Michael Luby

Qualcomm, Inc.
3195 Kifer Road
Santa Clara, CA 95051
USA
luby@qualcomm.com

the essence of knowledge

Boston – Delft

Foundations and Trends® in Communications and Information Theory

Published, sold and distributed by:
now Publishers Inc.
PO Box 1024
Hanover, MA 02339
USA
Tel. +1-781-985-4510
www.nowpublishers.com
sales@nowpublishers.com

Outside North America:
now Publishers Inc.
PO Box 179
2600 AD Delft
The Netherlands
Tel. +31-6-51115274

The preferred citation for this publication is A. Shokrollahi and M. Luby, Raptor Codes, Foundations and Trends® in Communications and Information Theory, vol 6, nos 3–4, pp 213–322, 2009

ISBN: 978-1-60198-446-3

Foundations and Trends® in Communications and Information Theory
Volume 6 Issues 3–4, 2009

Editorial Board

Editorial Scope

Foundations and Trends® in Communications and Information Theory will publish survey and tutorial articles in the following topics:

- Coded modulation
- Coding theory and practice
- Communication complexity
- Communication system design
- Cryptology and data security
- Data compression
- Data networks
- Demodulation and Equalization
- Denoising
- Detection and estimation
- Information theory and statistics
- Information theory and computer science
- Joint source/channel coding
- Modulation and signal design
- Multiuser detection
- Multiuser information theory
- Optical communication channels
- Pattern recognition and learning
- Quantization
- Quantum information processing
- Rate-distortion theory
- Shannon theory
- Signal processing for communications
- Source coding
- Storage and recording codes
- Speech and Image Compression
- Wireless Communications

Information for Librarians
Foundations and Trends® in Communications and Information Theory, 2009, Volume 6, 6 issues. ISSN paper version 1567-2190. ISSN online version 1567-2328. Also available as a combined paper and online subscription.

Foundations and Trends® in
Communications and Information Theory
Vol. 6, Nos. 3–4 (2009) 213–322

DOI: 10.1561/0100000060

now
the essence of knowledge

Raptor Codes

Amin Shokrollahi[1] and Michael Luby[2]

[1] *EPFL, Station 14, Lausanne 1015, Switzerland, amin.shokrollahi@epfl.ch*
[2] *Qualcomm, Inc., 3195 Kifer Road, Santa Clara, CA 95051, USA, luby@qualcomm.com*

Abstract

This monograph describes the theory behind Raptor codes, and elucidates elements of the processes behind the design of two of the most prominent members of this class of codes: R10 and RaptorQ (RQ). R10 has already been adopted by a number of standards' bodies, and RQ is in the process of entering various standards at the time of writing of this monograph.

The monograph starts with the description of some of the transmission problems, which inspired the invention of Fountain codes. Thereafter, Luby transform codes (LT codes) and Raptor codes are introduced and insights are provided into their design. These codes are currently the most efficient realizations of Fountain codes. Different algorithms are introduced for encoding and decoding various versions of these codes, including their systematic versions. Moreover, a hybrid decoding algorithm called "inactivation decoding" is introduced, which is an integral part of all modern implementations of Raptor codes.

The R10 and RQ codes have been continued and will continue to be adopted into a number of standards and thus there are publicly available specifications that describe exactly how to implement these

codes. However, the standards' specifications provide no insight into the rationale for the design choices made. One of the primary purposes of this document is to provide this design rationale.

We provide results of extensive simulations of R10 and RQ codes to show the behavior of these codes in many different scenarios.

Contents

1

Introduction

In theory, there is no difference between theory and practice, but in practice there is.[1]

This monograph describes the theory, design, and practical realization of Raptor codes. Section 2 describes conceptual design and analysis tools that provide provably good Raptor code designs. Section 3 describes more detailed design and heuristic analysis tools that provide constructions of practical Raptor codes. Based on their excellent recovery properties and computational complexity, these practical constructions have been standardized, implemented, and deployed.

In general and as seems to be universally typical, the performance of the practical constructions far exceed what can be rigorously proven. Although the theoretical tools and designs provide the big ideas and insights, it is the more detailed and precise heuristic tools and methods that have been developed and honed over the years to become precision instruments that were used to craft the design details of the

[1] Written on the interior glass wall of the EPFL cafeteria in the Computer Science Building BC, just near *Place Alan Turing*. Wikipedia attributed to Johannes L. A. van de Snepscheut and also to Yogi Berra.

practical Raptor codes, the R10 code and the RaptorQ (RQ) code. For the R10 and RQ codes, their provable properties are even more real than a theoretical proof: highly optimized software implementing the R10 and the RQ codes has been developed, tested, and deployed in mission critical applications. The R10 and RQ codes decoding properties have been verified again and again via tens of billions of simulations; their computational complexity has been verified again and again on different platforms with all kinds of parameter settings; their real-world practicality has been demonstrated by their adoption in a variety of commercial standards and by their deployment in commercial and government/defense markets. Thus, the R10 and RQ codes have achieved something much more valuable than just a theoretical existence proof: they have proven to be powerful, efficient, and flexible codes that provide a lot of practical value to a large variety of data communication applications.

1.1 Data Transmission

Digital media have become an integral part of modern lives. Whether surfing the web, making a wireless phone call, watching satellite TV, or listening to digital music, a large part of our professional and leisure time is filled with all things digital.

The replacement of analog media by their digital brethren and the explosion of the Internet use has had a perhaps unintended consequence. Whereas analog media were previously replaced by digital media to preserve quality, the existence of high-speed computer networks makes digital media available to potentially anyone, anywhere, and at any time. This possibility is the basis for modern scientific and economic developments centered around the distribution of digital data to a worldwide audience. The success of web sites such as Apple's iTunes store or YouTube is rooted in the marriage of digital data and the Internet.

Reliable transport of digital media to heterogeneous clients becomes thus a central and at time critical issue. Receivers can be anywhere and they may be connected to networks with widely differing fidelities.

1.2 The Transmission Control Protocol

How are data commonly transported on a network such as the Internet? The basic transmission protocol used by any Internet transmission is the Internet Protocol, commonly known as IP [4]. The data to be transmitted are subdivided into packets; these packets are given headers with information pertaining to their origin and their destination; pretty much like sending a regular letter, where we put the addresses of the receiver and that of the sender on the envelope. Routers that take the role of mail stations inspect these headers and forward the packets to another router closer to the destination. To do this, they consult regularly updated routing tables, through which they can determine the shortest path between them and the destination. Eventually, following the path from one router to another, packets may be delivered to their destinations.

In theory, this protocol is sufficient for data delivery; however, the reality looks different. Routers tend to get overwhelmed at times by incoming traffic, leading them to drop some of the incoming packets. These dropped packets will never reach their destination. To overcome this problem, researchers proposed already in the early days of the Internet the "Transmission Control Protocol," commonly known as TCP [5]. TCP has withstood the test of time, as it remains the most widely used transmission protocol on the Internet. For example, http (the protocol ubiquitously used for surfing the web), ssh (used for establishing a secure connection to a host), sftp (the secure file transfer protocol), and many other transmission protocols used today utilize TCP as a basis.

How does TCP work? We give here a very simplified description which has the advantage of clarifying the main mechanisms behind the real TCP. In effect, for every packet sent, an acknowledgment is expected from the receiver. If the acknowledgment is not received after a prescribed period of time, the packet is considered lost and counter mechanisms are initiated, with the most basic of these counter mechanisms consisting of resending the missing packet. The other integral part of these countermeasures is the reduction of the transmission rate, which is done in the following way: the real TCP does not await

acknowledgments of individual packets before sending the next one, but instead has at any time a number of packets in transit. The acknowledgment of a packet is only expected after all the packets sent previous to the packet have been acknowledged. When a packet is lost, detected at the server by received acknowledgments of packets sent after the lost packet, the number of packets allowed to be in transit is reduced, which effectively reduces the rate at which the packets are sent to the receiver and provides a rate control mechanism. The reason for this reduction in rate is the implicit assumption by TCP that losses have occurred because intermediate routers have been overwhelmed. The reduction of the sending rate is designed to reduce traffic on the routers and to alleviate the burden on the network.

1.3 The User Datagram Protocol

Another transmission protocol of interest to us is the "User Datagram Protocol" (UDP) [18]. Originally, this protocol was envisioned for short messages without strict reliability requirements. Essentially, it is analogous to sending mail through the postal service: each UDP packet contains a source address and a destination address, and the packet is routed through the network toward its destination without any guarantees that it will arrive; the packet may, e.g., be lost enroute due to a router buffer overflow, or to a wireless transmission error, etc. Furthermore, each UDP packet is sent independently of all other UDP packets, and a source may send a sequence of UDP packets at an arbitrarily high rate that can easily overload the network. Thus, UDP lacks TCP's higher-level reliability and rate control mechanisms. Because of this, delivery protocols that use UDP are sometimes blocked by firewalls from entering corporate networks.

Nevertheless, in some situations using UDP can be quite useful. For example, the destination address of a UDP packet can be a group address instead of an individual receiver address. Thus, any receiver that is part of the group can receive UDP packets sent to that group, thus enabling UDP to be used effectively in conjunction with a broadcast/multicast enabled network in a scalable way. For example, broadcast file delivery and broadcast streaming applications often use

UDP because the sent packets can be delivered concurrently to many receivers in a scalable way. In these types of applications, the packet sending rate is fixed at the source according to the available capacity of the network and/or the requirements of the application, e.g., real-time delivery of a 4 mbps video stream of packets. In these types of applications, adding a reliability protocol on top of UDP can be quite valuable, and providing this reliability is one of the main applications of Raptor codes.

1.4 Point-to-point Transmission

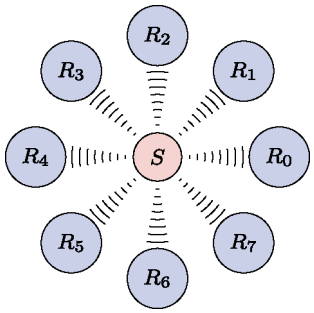
The simplest transmission scenario is point-to-point transmission. Here, a sender transmits data to one receiver, as described in the following figure in which a sender S is transmitting data to a receiver R:

If the distance between the sender and the receiver is not too large, then TCP is a perfect transmission protocol. However, if the distance is large, then TCP exhibits inefficiencies: during the time in which acknowledgments are awaited, transmission is in an idle mode and hence the real capacity of the network may not be achieved. The situation is compounded when there is loss on the network, i.e., the TCP transmission rate slows down even more.

1.5 Point-to-multipoint Transmission

The second transmission scenario is the point-to-multipoint transmission. The situation is described in the figure below, in which a sender S is transmitting data to receivers $R_0, \ldots, R_7$. A typical example is distributing live video over the Internet. Unless the number of receivers is small, TCP turns out to have some scaling issues in this setting. The reason is that the sender needs to keep track of the reception of every individual receiver, and furthermore that each receiver needs to be sent a separate stream of data. Therefore, the server load and the network load increase with the number of receivers, and reliable transmission

becomes more challenging. Ironically, the more popular the content, the more difficult it becomes to deliver it to all the receivers. This phenomenon that is typically referred to as the "curse of popularity" makes it difficult to provide a scalable broadcast service on the Internet. However, in recent years such services have started to be deployed, based on already deployed http caching server infrastructure.

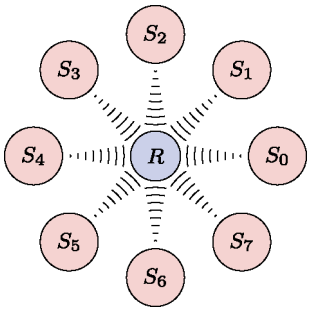


What are sometimes used for point-to-multipoint are protocols based on UDP, using the multicast/broadcast capabilities of UDP to handle the delivery scalability issue when the network is multicast/broadcast enabled, since all receivers in the destination group can attempt to receive a UDP packet sent to that destination group. However, as mentioned previously, UDP does not guarantee delivery of packets; it is a best effort protocol where sent packets may be lost for a variety of reasons, including wireless transmission noise that corrupts packets beyond repair, or because of packet overflows within the routers of the network due to intermittent congestion caused by other sources. Raptor codes can be used to provide reliability in a scalable way to UDP-based protocols.

1.6 Multipoint-to-point Transmission

Another scenario is multipoint-to-point transmission. Here, a group of senders, each possessing a copy of the same data, wants to transmit this copy to one receiver. The following figure shows an example in which senders $S_0, \ldots, S_7$ are transmitting to a common receiver R. In addition to problems discussed in the case of point-to-point transmission based on TCP, multipoint-to-point solutions based on TCP leads to enormous

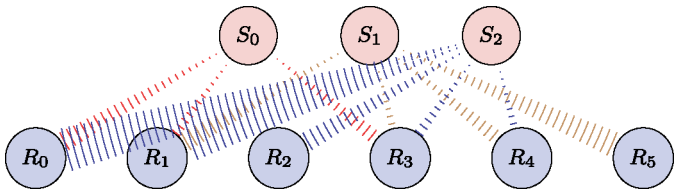
inefficiencies: the packets received from the various senders may not be different if the senders are not coordinated. The reception of duplicate packets is one of the principal sources of inefficient network usage in this case.



The fountain code properties (described in Section 1.8) make Raptor codes ideally suited for basing efficient solutions to the multipoint-to-point transmission protocols, where either UDP or TCP may be the underlying protocol onto which the usage of Raptor codes is layered.

1.7 Multipoint-to-multipoint Transmission

Another scenario is multipoint-to-multipoint transmission, depicted in the following figure in which we have a group of senders denoted S_0, S_1, S_2, each possessing a piece of data, and a group of receivers $R_0, \ldots, R_5$ each of which connects to a subset of the senders and receives the data:



A good example of this transmission scenario is a peer-to-peer network. All the problems discussed for the previous three transmission scenarios are also valid here. These problems are compounded if senders and receivers are transient, as is the case in a large peer-to-peer network. The fountain code properties (described in Section 1.8)

make Raptor codes ideally suited for usage in multipoint-to-multipoint scenarios.

1.8 Fountain Code Overview

At the very core of our solution lies the concept of a *fountain code*. We introduce the general use case for a fountain code, describe ideal abstract properties of a fountain code, describe its application to the scenarios described in the previous sections, and outline a randomized approach for constructing codes that lead toward the realization of practical fountain codes.

Suppose we have a block of data, hereafter called a *source block*, that are to be reliably transmitted over a packet network. The source block is typically partitioned into equal sized portions of data, hereafter called *source symbols*, that are typically sized to fit into a packet. In the following, we let k be the number of source symbols in the source block.

An effective approach to reliably transmitting the source block is to use an *encoder* at senders to generate *encoded symbols* from the source symbols of the source block and then to use *decoders* at receivers to decode the source symbols of the source block from received encoded symbols, where typically each encoded symbol is the same size as a source symbol. The basic idea is for senders to send encoded symbols in packets, and then a receiver can use the encoded symbols received in packets to try and decode the original source block even if some of the packets are not received. There are a variety of reasons a receiver may not receive some of the packets, examples of which include transmission over a wireless network that intermittently experiences enough interference or noise to cause unrecoverable errors in packets that are then discarded at the receiver, packet losses due to intermittent congestion that causes packet buffer overflows in routers, the receiver being only intermittently subscribed to the sessions in which packets are transmitted, and packets that arrive too late at the receiver to be useful when there are time constraints on consumption of the data in the source block.

Fountain codes in particular are especially effective codes that can be used to provide reliable transmission. The ideal abstract properties

of a fountain code are as follows [3]:

(1) A sender should be able to use a fountain encoder to generate as many encoded symbols as required from a source block.
(2) A receiver that receives any subset of k encoded symbols should (in most of the cases[2]) be able to use a fountain decoder to decode an exact copy of the original source block, independent of which subset of the generated encoded symbols is received and independent of whether the subset was generated by one sender or generated by more than one sender from the original block of source data.
(3) The computation time for encoding and decoding should be linear, i.e., the time to generate each encoded symbol should be linearly proportional to its size, and similarly the time to decode an original source block from encoded symbols should be linearly proportional to the original source block size.

These properties bring to mind a "fountain": When filling a bucket from a fountain of water, which particular drops fill the bucket doesn't matter, only the amount of water in the bucket matters. Similarly, with a fountain code, which particular encoded symbols are received doesn't matter, only the number of encoded symbols received matters.

From this description, it should be clear that fountain codes with these properties are very effective at providing reliable transmission over packet networks for any of the scenarios described in previous section. We now describe how fountain codes can be applied to these scenarios in a bit more detail.

In the point-to-point scenario, the sender can generate encoded symbols using a fountain encoder rom the source block and place the encoded symbols into packets, which are transmitted via the UDP protocol, for example. In a real-time application, the packets can be sent at any rate that is below the rate at which encoded symbols can be

[2] It is easily seen that a fountain code over a finite alphabet will not allow decoding from k received symbols in *all* the cases.

generated by the fountain encoder. Provided that this rate is very high, there will essentially be no limit on the transmission speed. Reliability of this transmission method is provided by the fountain property: as soon as the receiver collects k encoded symbols, it can decode the source symbols of the original source block. As k encoded symbols are the absolute bare minimum the receiver needs to collect to be able to decode the k source symbols, the transmission is optimal from an information point of view. The question of rate control remains, and in some cases it can be elegantly solved exploiting the fountain property [10, 11].

In the case of point-to-multipoint transmission, the sender generates encoded symbols and places them into packets and transmits the packets via, for example, broadcast or multicast. The fundamental properties of the fountain code guarantee that each receiver is capable of decoding the original data from reception of the minimal amount of data possible. Thus, one sender is capable of efficiently and reliably delivering to a potentially limitless number of receivers.

In the case of multipoint-to-point transmission, the various senders use fountain encoders applied to the common copy of the source block they each possess. The receiver collects encoded symbols from the various senders; by the properties of the fountain code, from the point of view of the receiver the mix of senders from which it receives encoded symbols does not matter. As soon as the receiver has collected k encoded symbols from the combined set of encoded symbols from the various senders, the original source block can be decoded.

The case of multipoint-to-multipoint transmission is solved in similar fashion and we will not elaborate further.

1.9 Fountain Code Construction Outline

Now that we know that fountain codes provide an elegant solution to various reliable transmission problem, we need to understand how to construct them. We now outline an approach that eventually leads to realizing almost ideal fountain codes. For a given vector $(x_1, \ldots, x_k)$ of source symbols, a fountain encoder produces a potentially limitless stream of encoded symbols $y_1, y_2, \ldots$. Here, a symbol refers to a bit or a sequence of bits. In many applications, symbols are of the same size as

the payload of the transmitted packets, though this is not necessarily the case. In general, the size of the symbols is often dictated by the underlying application and requirements.

The fountain codes that we initially describe operate on 1-bit symbols. Note that codes for larger symbols can be obtained using simple parallel concatenation, i.e., to generate a code that operates on t-bit symbols simply perform the same operations as would be performed on 1-bit symbols to each of the t positions of t-bit symbols in parallel.

These fountain codes are governed by a *probability distribution* $\mathcal{D}$ on the vector space $\mathbb{F}_2^k$. The encoding procedure for generating encoded symbol y_j is as follows:

(1) Sample $\mathcal{D}$ to obtain a vector $(a_1, \ldots, a_k) \in \mathbb{F}_2^k$.
(2) Calculate $y_j = \sum_i a_i x_i$.

The samplings of the fountain encoder are independent from encoded symbol to encoded symbol; this is extremely important as it induces a uniformity property on the encoded symbols generated and ensures that the code has the fountain properties. Note that when the encoded symbols are placed into packets for transmission, typically (but not always) an identifier is also placed in the header of each packet, called an ESI (encoded symbol identifier), that uniquely identifies the encoded symbols contained in that packet. The ESI is used by the decoder to determine the vector $(a_1, \ldots, a_k)$ corresponding to each encoded symbol in the received packet.

The average computational cost for generating an encoded symbol is simply the average weight of the vector $(a_1, \ldots, a_k) \in \mathbb{F}_2^k$ when sampled from $\mathcal{D}$ multiplied by the computational cost of adding two symbols together. Thus, it will be important to keep the average weight as small as possible.

Decoding algorithms will be described later; however, for now it is important to note that the following decoder performance metrics are key, and in particular the design of the probability distribution $\mathcal{D}$ has a large influence on these decoder performance metrics.

An important property we require of a fountain code is that it should be possible to decode the source symbols with little reception overhead with high probability. We say that the *overhead* is o if $k + o$ encoded

symbols are used when decoding is attempted, and if the overhead is written as a percent, i.e., $x\%$, then it is the overhead as a percent of the number of source symbols, i.e., $x = 100 \cdot o/k$.

The fountain code constructions we provide all have the property that encoded symbols are generated independently of one another. In addition, we will assume that the set of received encoded symbols is independent of the values of the encoded symbols in that set, an assumption that is often true in practice. These assumptions imply that for a given value of k, the probability of decoding failure is independent of the pattern of which encoded symbols are received and only depends on how many encoded symbols are received, i.e., the probability of decoding failure depends only on the overhead. Thus, we define the *failure probability* $f(o)$ to be the probability that decoding fails at a specified overhead o, i.e., the failure probability is a function of the overhead, and typically the failure probability should decrease quickly with increasing overhead. We call the set of pairs $\{(o, f(o)) : o = 0, 1, \ldots\}$ the *overhead-failure curve*.

Often we analyze the failure probability at a small overhead of interest as a function of k, i.e., for an overhead of $\varepsilon(k)$ we provide upper bounds on $f(\varepsilon(k))$, where both $\varepsilon(k)/k$ and $f(\varepsilon(k))$ go to zero as k goes to infinity. For example, it might be the case that $\varepsilon(k) = \varepsilon \cdot k$ for some constant ε, $0 < \varepsilon \ll 1$, and $f(\varepsilon \cdot k) = 1/k^c$ for some positive constant c, where preferably $c > 1$. In practice, what is important is that the failure probability decreases as quickly as possible as a function of increasing overhead, i.e., the overhead-failure curve is steep.

Equally important, the decoder should be computationally very efficient.

1.10 The Random Binary Fountain

A natural fountain code to consider is the "random binary fountain code," where the distribution $\mathcal{D}$ is the uniform distribution on $\mathbb{F}_2^k$, where k is the number of 1-bit (binary) source symbols in the source block. As described previously, this can be extended to symbols of arbitrary size.

Let us give a qualitative analysis of a random binary fountain code. The receiver collects $N = k + o$ encoded symbols $y_1, y_2, \ldots, y_N$. Each

of these symbols is a uniform random linear combination of the source symbols $x_1, \ldots, x_k$. The relationship between the source and the collected encoded symbols is described by a matrix, $A \in \mathbb{F}_2^{N \times k}$, as

$$A \cdot \begin{pmatrix} x_1 \\ \vdots \\ x_k \end{pmatrix} = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_N \end{pmatrix}.$$

This matrix A is chosen uniformly at random from the set of binary $N \times k$ matrices.

Recovery of the source symbols is possible iff the rank of A is k. A simple analysis [20, Proposition 2] shows the following result:

Proposition 1.1. For a random binary fountain code operating on a source block with k source symbols, the overhead-failure curve is pointwise majorized by $\{(o, 2^{-o}) : o = 0, 1, \ldots\}$ with respect to the maximum-likelihood decoder.

For example, at an overhead of c-$\log_2(k)$, the failure probability is $1/k^c$. In fact, one can show that for not too small values of o, $f(o)$ is roughly 2^{-o}.

As the above proposition describes, a random binary fountain code has a quickly decreasing failure probability as a function of overhead, i.e., the failure probability decreases by almost exactly a factor of two for each increase by one in the overhead. For example, a failure probability of 10^{-10} can be achieved with an overhead of around 30 symbols, regardless of k. For moderate values of k, say in the low thousands, this overhead relative to k is smaller than 0.3%.

However, random binary fountain codes suffer from a large encoding and decoding computational complexity. To assess this complexity, we will distinguish between "symbol operations" and "bit operations." The former corresponds to XORs of symbols, whereas the latter corresponds to XORs of bits. When the symbol size is large, a symbol size operation may be significantly more computationally expensive than a bit operation.

On average, every encoded symbol will be the XOR of around half the source symbols; hence, take around $k/2$ symbol operations to be created.[3]

The decoding takes $O(k^3)$ bit operations and $O(k^2)$ symbol operations. To prove this, we proceed as follows: first, we determine a $k \times k$-submatrix B of A, which is invertible over $\mathbb{F}_2$, and we determine its inverse B^{-1}. This can be done using the Gaussian elimination, and requires $O(k^3)$ bit operations.[4] The matrix B is determined by k rows of A, say rows $1, \ldots, k$. Next, we multiply B^{-1} with the vector consisting of $y_1, \ldots, y_k$. As B^{-1} has $O(k^2)$ entries equal to 1, the number of symbol operations is $O(k^2)$.

Summarizing, the random binary fountain code achieves a good overhead-failure curve; however, both encoding and decoding are computationally complex. What we would like instead is a fountain code that achieves similar or even an improved overhead-failure curve and has computationally efficient encoding and decoding algorithms.

For future use in Section 3.3.1, we mention that the concept of a random fountain is not limited to the field $\mathbb{F}_2$. More generally, we talk about a "q-ary random fountain code," or a "fountain code over $\mathbb{F}_q$" if the distribution $\mathcal{D}$ is chosen to be uniform over $\mathbb{F}_q^k$.

[3] This is not full proof; it is conceivable that a faster algorithm is available than simply XORing all the corresponding source symbols. Because of the random structure of the random binary fountain code, this seems highly unlikely though.

[4] There are faster algorithms based on fast matrix multiplication; however, they are not practically relevant.

References

[1] 3GPP TS 26.346 V6.1.0, "*Technical Specification Group Services and System Aspects; Multimedia Broadcast/Multicast Service; Protocols and Codecs*," June, 2005.

[2] J. R. Bitner, G. Ehrlich, and E. M. Reingold, "Efficient generation of the binary reflected Gray code and its applications," *Communications of the ACM*, vol. 19, no. 9, pp. 517–521, 1976.

[3] J. Byers, M. Luby, M. Mitzenmacher, and A. Rege, "A digital fountain approach to reliable distribution of bulk data," in *Proceedings of ACM SIGCOMM '98*, 1998.

[4] DARPA Internet Program, "Internet protocol," September 1981, Internet Engineering Task Force, RFC 791. Available at http://www.ietf.org/rfc/rfc0793.txt?number=791.

[5] DARPA Internet Program, "Transmission control protocol," September 1981, Internet Engineering Task Force, RFC 793. Available at http://www.ietf.org/rfc/rfc0793.txt?number=793.

[6] ETSI TS 102 472 v1.2.1, "*IP Datacast over DVB-H: Content Delivery Protocols*," March 2006, Technical Specification. Available at http://www.dvb-h.org.

[7] R. G. Gallager, *Low Density Parity-Check Codes*. Cambridge, MA: MIT Press, 1963.

[8] B. A. Lamacchia and A. M. Odlyzko, "Solving large sparse linear systems over finite fields," in *Proceedings CRYPTO'90*, pp. 109–133, Springer, 1991.

[9] M. Luby, "LT-codes," in *Proceedings of the 43rd Annual IEEE Symposium on the Foundations of Computer Science (FOCS)*, pp. 271–280, 2002.

[10] M. Luby and V. Goyal, "Wave and equation based rate control," April 2004, Internet Engineering Task Force, RFC 3738. Available at http://tools.ietf.org/html/rfc3738.

[11] M. Luby, V. Goyal, S. Skaria, and G. Horn, "Wave and equation based rate control," in *Proceedings of SIGCOMM*, pp. 191–204, 2002.

[12] M. Luby, M. Mitzenmacher, and A. Shokrollahi, "Analysis of random processes via and-or tree evaluation," in *Proceedings of the 9th Annual ACM-SIAM Symposium on Discrete Algorithms*, pp. 364–373, 1998.

[13] M. Luby, M. Mitzenmacher, A. Shokrollahi, and D. Spielman, "Efficient erasure correcting codes," *IEEE Transactions Information Theory*, vol. 47, pp. 569–584, 2001.

[14] M. Luby, M. Mitzenmacher, A. Shokrollahi, D. Spielman, and V. Stemann, "Practical loss-resilient codes," in *Proceedings of the 29th Annual ACM Symposium on Theory of Computing*, pp. 150–159, 1997.

[15] M. Luby, A. Shokrollahi, M. Watson, and T. Stockhammer, "Raptor forward error correction scheme for object delivery," September 2007, Internet Engineering Task Force, RFC 5053. Available at http://tools.ietf.org/html/rfc5053.

[16] M. Luby, A. Shokrollahi, M. Watson, T. Stockhammer, and L. Minder, "RaptorQ forward error correction scheme for object delivery," August 2010, Internet Engineering Task Force. Available at http://tools.ietf.org/html/draft-ietf-rmt-bb-fec-raptorq-03.

[17] C. Pomerance and J. W. Smith, "Reduction of huge, sparse matrices over finite fields via created catastrophes," *Experimental Math*, vol. 1, pp. 89–94, 1992.

[18] J. Postel, "User datagram protocol," August 1980,Internet Engineering Task Force, RFC 768. Available at http://www.ietf.org/rfc/rfc0768.txt?number=768.

[19] T. Richardson and R. Urbanke, "Efficient encoding of low-density parity-check codes," *IEEE Transactions Information Theory*, vol. 47, pp. 638–656, 2001.

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

[21] A. Shokrollahi, "Theory and applications of raptor codes," in *Proceedings of MathKnow*, pp. 59–89, 2009.

[22] A. Shokrollahi, S. Lassen, and R. Karp, "Systems and processes for decoding chain reaction codes through inactivation," U.S. Patent number 6,856,263. February 15, 2005.

[23] A. Shokrollahi, S. Lassen, and M. Luby, "Multi-stage code generator and decoder for communication systems," U.S. Patent 7,068,729. June 27, 2006.

[24] A. Shokrollahi and M. Luby, "Systematic encoding and decoding of chain reaction codes," U.S. Patent 6 909 383. June 21, 2005.

[25] V. V. Zyablov and M. S. Pinsker, "Decoding complexity of low-density codes for transmission in a channel with erasures," *Probl. Information Transmission*, vol. 10, 1974.