1 Introduction

We recall an example application in which millions of clients want to download a new release of software over the course of several days. In this application, we assume that there is a distribution server, and that the server will send out a stream of packets (using either broadcast or multicast) as long as there are clients attempting to download the new release. This software download application highlights several important features common to many similar applications that must distribute bulk data. In addition to keeping network traffic to a minimum, a scalable protocol for distributing the software using multicast should be: • Reliable: The file is guaranteed to be delivered in its entirety to all receivers. • Efficient: Both the total number of packets each client needs to receive and the amount of time required to process the received packets to reconstruct the file should be minimal. Ideally, the total time for the download for each client should be no more than it would be had point-to-point connections been used. • On demand: Clients may initiate the download at their discretion, implying that different clients may start the download at widely disparate times. Clients may sporadically be interrupted and continue the download at a later time. • Tolerant: The protocol should tolerate a heterogeneous population of receivers, especially a variety of end-to-end packet loss rates and data rates. We also state our assumptions regarding channel characteristics. IP multicast on the Internet, satellite transmission, wireless transmission, and cable transmission are representative of channels we consider. Perhaps the most important property of these channels is that the return feedback channel from the clients to the server is typically of limited capacity, or is non-existent

2 The digital fountain

A server wishes to allow a universe of clients to acquire source data consisting of a sequence of k equal length packets. In the idealized solution, the server sends out a stream of distinct packets, called encoding packets, that constitute an encoding of the source data. The server will transmit the encoding packets whenever there are any clients listening in on the session. A client accepts encoding packets from the channel until it obtains exactly k packets. In this idealized solution, the data can be reconstructed regardless of which k encoding packets the client obtains. Therefore, once k encoding packets have been received the client can disconnect from the channel. We assume that in this idealized solution that there is very little processing required by the server to produce the encoding of packets and by the clients to recover the original data from k encoding packets. We metaphorically describe the stream of encoding packets produced by the server in this idealized solution as a digital fountain.

2.1 Erasure codes

An ideal way to implement a digital fountain is to directly use an erasure code that takes source data consisting of k packets and produces sufficiently many encoding packets to meet user demand. Indeed, standard erasure codes such as Reed-Solomon erasure codes have the ideal property that a decoder at the client side can reconstruct the original source data whenever it receives any k of the transmitted packets. But erasure codes are typically used to stretch a file consisting of k packets into n encoding packets, where both k and n are input parameters. We refer to the ratio n/k as the stretch factor of an erasure code. While this finite stretch factor limits the extent to which erasure codes can approximate a digital fountain, a reasonable approximation proposed by other researchers (e.g., [18, 22, 23, 25]), is to set n to be a multiple of k, then repeatedly cycle through transmission of the n encoding packets. The limitation is that for any pre-specified value of n, under sufficiently high loss rates a client may not receive k out of n packets in one cycle. Thus in lossy environments, a client may receive useless duplicate transmissions before reconstructing the source data, decreasing the channel efficiency. But in practice, our experimental results indicate that this source of inefficiency is not large

even under very high loss rates and when n is set to be a small multiple of k, such as n = 2k, the setting we use in the remainder of the paper. A more serious limitation regards the efficiency of encoding and decoding operations. As detailed in subsequent sections, the encoding and decoding processing times for standard Reed-Solomon erasure codes are prohibitive even for moderate values of k and n. The alternative we propose is to avoid this cost by using the much faster Tornado codes [11]. As always, there is a tradeoff associated with using one code in place of another. The main drawback of using Tornado codes is that the decoder requires slightly more than k of the transmitted packets to reconstruct the source data. This tradeoff is the main focus of our comparative simulation studies that we present in Section 6. But first, in Section 5, we provide an in-depth description of the way Tornado codes are constructed and their properties.

2.2 Tornado codes

construction of a specific Tornado code and explain some of the general principles behind Tornado codes. We first outline how these codes differ from traditional Reed- Solomon erasure codes. Then we give a specific example of a Tornado code based on [11, 12] and compare its performance to a standard Reed-Solomon code. For the rest of the discussion, we will consider erasure codes that take a set of k source data packets and produce a set of ToBeInsterted redundant packets for a total of n = k + ToBeInsterted encoding packets all of a fixed length P. 5.1 Theory We begin by providing intuition behind Reed-Solomon codes. We think of the ith source data packet as containing the value of a variable xi, and we think of the jth redundant packet as containing the value of a variable yj that is a linear combination of the xi variables over an appropriate finite field. (For ease of description, we associate each variable with the data from a single packet, although in our simulations each packet may hold values for several variables.) For example, the third redundant packet might hold y3 = x1 +x2ToBeInsterted+. . .+xkToBeInstertedkToBeInsterted1, where ToBeInsterted is some primitive element of the field. Typically, the finite field multiplication operations are implemented using table lookup and the addition operations are implemented using exclusive-or. Each time a packet arrives, it is equivalent to receiving the value of one of these variables. Reed-Solomon codes guarantee that successful receipt of any k distinct packets enables reconstruction of the source data. When e redundant packets and k ToBeInsterted e source data packets arrive, there is a system of e equations corresponding to the e redundant packets received. Substituting all values corresponding to the k received packets into these equations takes time proportional to (k ToBeInsterted e + 1)e. The remaining subsystem has e equations and e unknowns corresponding to the source data packets not received. With Reed-Solomon codes, this system has a special form that allows one to solve for the unknowns in time proportional to e2 via a matrix inversion and matrix multiplication. The large decoding time for Reed-Solomon codes arises from the dense system of linear equations used. Tornado codes are built using random equations that are sparse, i.e. the average number of variables per equation is small. This sparsity allows substantially more efficient encoding and decoding. The price we pay for much faster encoding and decoding is that k packets no longer suffice to reconstruct the source data; instead slightly more than k packets are needed. In fact, designing the proper structure for the system of equations so that the number of additional packets and the coding times are simultaneously small is a difficult challenge [11, 12]. For Tornado codes, the equations have the form y3 = x1 ToBeInsterted x4 ToBeInsterted x7, where ToBeInsterted is bitwise exclusive-or. Tornado codes also use equations of the form y53 = y3ToBeInstertedy7ToBeInstertedy13; that is, redundant packets may be derived from other redundant packets. The encoding time is dominated by the number of exclusive-or operations in the system of equations. The decoding process for Tornado codes uses two basic operations. The first operation consists of replacing the received variables by their values in the equations in which they appear. The second

operation is a simple substitution rule. The substitution rule can be applied to recover any missing variable that appears in an equation in which that variable is the unique missing variable. For example, consider again the equation y3 = x1 ToBeInsterted x4 ToBeInsterted x7. Suppose the redundant packet containing y3 has been received, as well as the source data packets containing x1 and x7, but x4 has not been received. Then we can use the above equation to solve for x4, again using only exclusive-or operations. Using this substitution rule repeatedly, a single packet arrival may allow us to reconstitute several additional packets, as the effect of that arrival propagates. In practice, the number of possible substitution rule applications remains minimal until slightly more than k packets have arrived. Then often a single arrival generates a whirlwind of substitutions that allow recovery of all of the remaining source data packets. Hence the name Tornado codes. The decoding may stop as soon as enough packets arrive so that the source data can be reconstructed. Note that Tornado codes use only exclusive-or operations and avoid both the field operations and the matrix inversion inherent in decoding Reed-Solomon codes. The total number of exclusive-or operations for decoding is at most the number used for encoding, and in general is less. As we have stated, to reconstruct the source data using a Tornado code, it suffices to recover slightly more than k of the n packets. We say that the decoding inefficiency is 1+ if (1+)k encoding packets are required to reconstruct the source data. For Tornado codes the decoding inefficiency is not a fixed quantity but depends on the packet loss pattern and the random choices used to construct the code. This variance in decoding inefficiency is described in more detail in Section 5.3. The advantage of Tornado codes over standard codes is that they trade off a small increase in decoding inefficiency for a substantial decrease in encoding and decoding times.

2.3 Simulation comparisons

From the previous section, it is clear that using Reed- Solomon erasure codes to encode over large files for bulk data distribution has prohibitive encoding and decoding overhead. But another approach, described in the introduction, is the method of interleaving suggested in [20, 22, 23, 24]. Interleaved codes are constructed as follows: suppose K + L encoding packets are to be produced from K file packets. Partition the K file packets into blocks of length k, so that there are B = K/k blocks in total. Stretch each block of k packets to an encoding block of k + ToBeInsterted packets using a standard erasure code by adding ToBeInsterted = kL/K redundant packets. Then, form the encoding of length K + L by interleaving the encoding packets from each block, i.e., the encoding consists of sequences of B packets, each of which consist of exactly one packet from each block. The choice of the value of the parameter k for interleaved codes is crucial. To optimize encoding and decoding speed of the interleaved codes, k should clearly be chosen to be as small as possible. But choosing k to be very small defeats the purpose of using encoding, since any redundant packet that arrives can only be used to reconstruct a source data packet from the same block. Moreover, redundant packets that arrive for data blocks that have already been reconstructed successfully do not benefit the sender. To explain this in more detail, let us say that a block is full from the client viewpoint when at least k distinct transmitted packets associated with that block have been received. The entire file can only be decoded by the client when all blocks are full. (Note however that some of the decoding work can potentially be done in the background while packets arrive; the same also holds for Tornado codes.) The phenomenon that arises when k is relatively small is illustrated in Figure 3; while waiting for the last few blocks to fill, the receiver may receive many packets from blocks that have already been reconstructed successfully. These useless packets contribute directly to the decoding inefficiency. To summarize, the choice of the value of k for interleaved codes introduces a tradeoff between decoding speed and decoding inefficiency. To compare various protocols, we compare the decoding inefficiency and decoding speed at each receiver. Recall that the decoding inefficiency

is 1+ if one must obtain (1 +)k distinct packets in order to decode the source data. For Tornado codes, there is some decoding inefficiency based on how the codes are constructed. For interleaved codes, decoding inefficiency arises because in practice one must obtain more than k packets to have enough packets to decode each block. We emphasize that for interleaved codes the decoding inefficiency is a random variable that depends on the loss rate, loss pattern, and the block size. The tradeoff between decoding inefficiency and coding time for interleaved codes motivates the following set of experiments. • Suppose we choose k in the interleaved setting so that the decoding inefficiency is comparable to that of Tornado Z. How does the decoding time compare? • Suppose we choose k in the interleaved setting so that the decoding time is comparable to that of Tornado Z. How does the decoding inefficiency compare? In our initial simulations, we assume probabilistic loss patterns in which each transmission to each receiver is lost independently with a fixed probability p. We emphasize that using bursty loss models instead of this uniform loss model would not impact our results for Tornado code performance; only the overall loss rate is important. This is because when using Tornado codes we compute the entire encoding ahead of time and send out packets in a random order from the source end. Therefore, any loss pattern appears equivalent to a uniform loss pattern on the receiver end. Note that this randomization at the sender end may introduce latency, and therefore this Tornado code approach may not be appropriate for some applications such as real-time interactive video. The choice of the uniform model does however impact the performance results of the interleaved codes, which (unless the same randomization of the transmission order is used) are highly dependent on the loss pattern. In particular, we would expect interleaved codes to have slightly better performance under bursty losses. We therefore also provide results from tracedriven simulations of the Internet to demonstrate the relatively small effect of burstiness on interleaved code performance. 6.1 Equating Decoding Efficiency Our first simulation compares the decoding time of Tornado Z with an interleaved code with decoding inefficiency comparable to those of Tornado Z. In Section 5, we determined experimentally that Tornado Z codes have the property that the decoding inefficiency is greater than 1.076 less than 1present the ratio between the running time of an interleaved code for which k is chosen so that this property is also realized and the running time of Tornado Z. Of course, this ratio changes as the loss probability and file size change. We explain how the entries in Table 4 are derived. To compute the running time for interleaved codes, we Speedup factor for Tornado Z erasure probabilities SIZE 0.01 0.05 0.10 0.20 0.50 250 KB 1.37 2.05 5.55 11.1 11.1 500 KB 2.29 5.51 8.33 16.7 33.3 1 MB 4.12 10.3 17.1 25.8 51.6 2 MB 6.34 16.9 26.2 48.4 96.8 4 MB 7.87 22.3 34.6 62.7 115 8 MB 11.1 28.2 46.9 80 182 16 MB 14.2 34.9 56.4 100 212 Table 4: Speedup of Tornado Z codes over interleaved codes with comparable efficiency. first use simulations to determine for each loss probability value the maximum number of blocks the source data can be split into while still maintaining a decoding inefficiency less than 1.076 for less than 1(For example, a two megabyte file consisting of 2000 one kilobyte packets can be split into at most eleven blocks while maintaining this property when packets are lost with probability 0.10.) We then calculate the decoding time per block, and multiply by the number of blocks to obtain the decoding time for the interleaved code. With a stretch factor of two, one half of all packets injected into the system are redundant encoding packets and the other half are source data packets. Therefore, in computing the decoding time per block, we assume that half the packets received are redundant encoding packets. Based on the data previously presented in the Cauchy codes column of Table 3, we approximate the decoding time for a block of k source data packets by k2/31250 seconds. To compute the running time for Tornado Z, we simply use the decode times for Tornado Z as given earlier in Table 3. As an example, suppose the encoding of a 16 MB file is transmitted over a 1 Mbit/second channel with a loss rate of 50enough packets to decode the file using either Tornado Z or an interleaved code

(with the desired decoding inefficiency guarantee), but then the decoding time is almost 8 minutes for the interleaved code compared with just over 2 seconds for Tornado Z. Comparisons for encoding times yield similar results. We note that by using slightly slower Tornado codes with less decoding inefficiency, we would actually obtain even better speedup results at high loss rates. This is because interleaved codes would be harder pressed to match stronger decoding guarantees.that have comparable decoding times to Tornado Z. Cauchy codes with block length k = 20 are roughly equivalent in speed to the Tornado Z code. We also compare with a block length k = 50, which is slower but still reasonable in practice. Using these block sizes, we now study the maximum decoding inefficiency observed as we scale to a large number of receivers. The sender carousels through a two megabyte encoding of a one megabyte file, while receivers asynchronously attempt to download it. We simulate results for the case in which packets are lost independently and uniformly at random at each receiver at rates of 10of congested Internet connections, while the 50receiver with poor connectivity might reasonably experience. The results we give can be interpolated to provide intuition for performance at intermediate rates of loss. For channels with very low loss rates, such as the 1Tornado have generally comparable performance. Figure 4 shows for different numbers of receivers the worst case decoding efficiency experienced for any of the receivers averaged over 100 trials. In these figures, p refers to the probability a packet is lost at each receiver. Since the leftmost point in each subfigure is for the case of one receiver, this point is also just the average decoding inefficiency. The interesting feature of this figure is how the worst case decoding inefficiency grows with the number of receivers. For packet loss rates of 10k = 50, the average inefficiency of interleaved codes is comparable to that of Tornado Z. But as packet loss rates increase, or if a smaller block size is used, the inefficiency of interleaved codes rises dramatically. Also, the inefficiency of the worst-case receiver does not scale with interleaved codes as the receiver size grows large. Tornado codes exhibit more robust scalability and better tolerance for high loss rates. 6.3 Scaling to Large Files Our next experiments demonstrate that Tornado codes also scale better than an interleaved approach as the file size grows large. This is due to the fact that the number of packets a client must receive to reconstruct the source data when using interleaving grows super-linearly in the size of the source data. (This is the well-known "coupon collector's problem.") In contrast, the number of packets the receivers require to reconstruct the source data using Tornado codes grows linearly in the size of the source data, and in particular the decoding inefficiency does not increase as the file size increases. The effect of this difference is easily seen in Figure 5. In this case both the average decoding inefficiency and the maximum decoding inefficiency grow with the length of the file when using the interleaving. This effect is completely avoided by using Tornado codes. 6.4 Trace-Driven Simulations To study the impact of bursty loss patterns on the relative performance of Reed-Solomon and Tornado code approaches, we perform a similar comparison using publicly available MBone trace data collected by Yajnik, Kurose, and Towsley [26]. In these traces, between six and twenty clients from the US and abroad subscribed to MBone broadcasts each of roughly an hour in length and reported which packets they received. Clients experienced packet loss rates ranging from less than 1to well over 20To sample loss patterns from these traces, we simply chose a random starting point for each broadcast, and then used the trace data to generate packet loss patterns for each receiver in the broadcast beginning at that time. We then simulated downloading files of various lengths using interleaving and using Tornado codes with these loss patterns. Averaging over 146 loss patterns generated from 15 broadcasts, we plot the average decoding inefficiency for various file sizes in Figure 6. The average loss rate over the randomly chosen trace segments we selected was just over 11data, there was considerable variance in the loss rate; some clients received virtually every packet, others experienced large burst losses over significant periods of time. While this trace data is limited

in scope, the Tornado codes maintain superior decoding inefficiency in the presence of high burst losses present in this data set. In fact, the results appear very similar to that in Figure 5 when p = 0.1, suggesting that the bursty loss pattern has only a small effect.

3 The LT-codes

LT codes are the first realization of a class of erasure codes that we call universal erasure codes. The symbol length for the codes can be arbitrary, from one-bit binary symbols to general ToBeInsterted-bit symbols. We analyze the run time of the encoder and decoder in terms of symbol operations, where a symbol operation is either an exclusive-or of one symbol into another or a copy of one symbol to another. If the original data consists of k input symbols then each encoding symbol can be generated, independently of all other encoding symbols, on average by O(ln(k/ToBeInsterted)) symbol operations, and the k original input symbols can be recovered from any k+O(ToBeInsterted k ln2(k/ToBeInsterted)) of the encoding symbols with probability 1 ToBeInsterted ToBeInsterted by on average O(k · ln(k/ToBeInsterted)) symbol operations. LT codes are rateless, i.e., the number of encoding symbols that can be generated from the data is potentially limitless. Furthermore, encoding symbols can be generated on the fly, as few or as many as needed. Also, the decoder can recover an exact copy of the data from any set of the generated encoding symbols that in aggregate are only slightly longer in length than the data. Thus, no matter what the loss model is on the erasure channel, encoding symbols can be generated as needed and sent over the erasure channel until a sufficient number have arrived at the decoder in order to recover the data. Since the decoder can recover the data from nearly the minimal number of encoding symbols possible, this implies that LT codes are near optimal with respect to any erasure channel. Furthermore, the encoding and decoding times are asymptotically very efficient as a function of the data length. Thus, LT codes are universal in the sense that they are simultaneously near optimal for

3.1 Encoding process

The process of generating an encoding symbol is conceptually very easy to describe: • Randomly choose the degree d of the encoding symbol from a degree distribution. The design and analysis of a good degree distribution is a primary focus of the remainder of this paper. • Choose uniformly at random d distinct input symbols as neighbors of the encoding symbol. • The value of the encoding symbol is the exclusive-or of the d neighbors.When using the encoding symbols to recover the original input symbols of the data, the decoder needs to knowthe degree and set of neighbors of each encoding symbol. There are many different ways of communicating this information to the decoder, depending on the application. For example, the degree and a list of neighbor indices may be given explicitly to the decoder for each encoding symbol.

3.2 The degree distribution, a quick list

Definition 3 (degree distribution): For all d, ToBeInsterted(d) is the probability that an encoding symbol has degree d. **(Cri says: Ora contestualizza le degree distr e abbrevia questo:)** As we now develop, the random behavior of the LT process is completely determined by the degree distribution ToBeInsterted(·), the number of encoding symbols K, and the number of input symbols k. Our objective is to design degree distributions that meet the following two design goals. • As few encoding symbols as possible on average are required to ensure success of the LT process. Recall that the number of encoding symbols that ensure success of the LT process corresponds to the number of encoding symbols that ensure complete recovery of the data. • The average degree of the encoding symbols is as low as possible. The average degree is the number of symbol operations on average it takes to generate an encoding symbol. The average degree timesK

is the number of symbol operations on average it takes to recover the entire data.

- all at once etc
- Ideal soliton Stretc
- Rosbust soliton

3.3 The decoding process

Definition 1 (decoder recovery rule): If there is at least one encoding symbol that has exactly one neighbor then the neighbor can be recovered immediately since it is a copy of the encoding symbol. The value of the recovered input symbol is exclusive-ored into any remaining encoding symbols that also have that input symbol as a neighbor, the recovered input symbol is removed as a neighbor from each of these encoding symbols and the degree of each such encoding symbol is decreased by one to reflect this removal. **NOTE:** as an introduction

(Cri says: For general finite length analysis) In paper 9 the authors present breifly an efficient method to analyse the error probability of Belief Propagation (BP) decoder applied to this type of codes.To be specific, **(Cri says: continua da qui)**
At each point in the decoding process of an LT Code the state of the decoder is described by three parameters: The number u of input symbols that are still unrecovered, the number r of output symbols of degree one at this point (the output ripple), and the number c of output symbols whose degree at this point is larger than 1 (the cloud). For fixed degree distribution, number k of input symbols, and number n of collected output symbols, the decoder will be in a state (c, r, u) with a certain probability Pc,r,u. Let Pu(x, y) := P c,r,ToBeInsterted Pc,r,uxcyr-1 be the generating function of this probability distribution given that exactly u input symbols are undecoded. Then we have the following. Theorem 1. Suppose that the original code has k input symbols and that n = k(1 + ToBeInsterted) output symbols have been collected for decoding. Further, denote by ToBeInsterted i, i = 1, . . . , D, the probability that an output symbol is of degree i, where D

3.3.1 In the case of small length

A. LT codes LT codes proposed by Luby in [5] are the first codes fully realizing the digital fountain concept presented in [1]. They are rateless, i.e., the rate does not need to be fixed beforehand, and encoded symbols are generated on the fly [6], [7].If the degree of the packet is still higher than 1 it consists of several original blocks and it is stored in a buffer. If a degree- 1 packet is recovered, it is identical to an original block, i.e., a new block has been discovered. Next the newly discovered block is removed from the other buffered packets including it. If this step reveals new degree-1 packets, the decoding continues iteratively until the original message is fully decoded. Otherwise the decoder has to wait for a new packet. The decoding process is sketched in listing Algorithm 2. B. Notation We denote the number of blocks (or input symbols) in the message by N and the degree distribution by ToBeInsterted(d). When convenient, we also refer to the point probabilities by pj , i.e., pj = ToBeInsterted(j). Later we will compare the performance of the optimized distributions to the following reference distributions: Def.1 (Uniform): pi = 1/N, i = 1, . . . , N. Def.2 (Degree-1): pi = 1(i = 1). Def.3 (Binomial): pi= 1 2N-1 ToBeInserted N i ToBeInserted , i=1, . . . , N. Def.4 (Soliton): p1= 1 N , and pi= 1 i(i-1), i=2, . . . , N. Binomial distribution is the standard Bin(N, 1 2) with event 0 excluded. It results from including every source block independently with probability 1 2 , discarding an empty packet. Let the random variable Zk denote the number of decoded input symbols after receiving the kth packet. Initially, Z0 = 0 and at the end Zk = N. The random variable T denotes the number of packets needed for decoding the original message, T = min k k : Zk = N. The earliest time when the decoding process can finish is when the Nth packet arrives and thus T ToBeInserted N. Moreover, we let PN denote the probability that a message consisting of N blocks is successfully decoded with exactly N received packets, PN = PZN = N = PT = N . **(Cri says: "So in the paper two ways of defining a probability are**

presented:")

- The decoding process can be studied as a Markov chain [8]. From the receiver's point of view, the set of received and either partially or fully decoded packets denotes a state. State transition probabilities depend on the arrival probabilities of specific packets, which in turn depend on the degree distribution used in the encoding. The process ends when it has reached the absorbing state consisting of the original blocks. The state space of the Markov chain describing the decoding process, however, needs only include the states that are irreducible in the sense that they cannot be reduced by the decoder. For instance the raw state a, abc is not included as decoding reduces it to the state a, bc. This decreases the number of states remarkably. Further reduction is possible by observing that if a sample path of the process is modified by permuting the original blocks then the resulting sample path is isomorphic to the original one. Using the above reduction schemes the number of states can be brought down to 12 for N=3 and to 192 for N=4. Systems of this size are amenable to numerical analysis. For N=5, however, even the reduced state space has 612224 states, which is too much to be conveniently handled, and for N ToBeInserted 5 the task is overwhelming. We consider two different optimization criteria for the degree distribution. A natural objective is to minimize the mean number of packets needed to successfully decode the message. Alternatively one may wish to maximize the probability of successful decoding after reception of N packets. Using the state transition matrix P we can calculate the average number of sent packets needed to recover the whole original file. This Markov chain has now the state where all blocks are decoded as an absorbing state. In the example with three blocks, using the notation presented, this state is a, b, c. As this Markov chain is clearly finite, it can be written in the following canonical form: P = ToBeInserted Q R 0 I ToBeInserted , where Q is the transition matrix between transient states, R represents transitions from transient states to absorbing ones and I is identity matrix corresponding to the absorbing states. In our case, there is just one absorbing state and the identity matrix I reduces to a 1x1 matrix. Now the fundamental matrix M = (I - Q)-1 is well-defined with all elements positive and represents all possible transition sequences in the transient states without going to the absorbing one. A specific element mij in M tells the mean number of visits in state j before absorption when starting in state i. Using the fundamental matrix, average number of steps can be calculated as follows abs, From this result it is easy to calculate the optimal weights minimizing the mean number of steps to decode the message. Similarly, using (2) one obtains an expression for P3, the probability of full decoding after 3 received packets, identical with (5) obtained by the approach of Section IV. Optimized results for these two different objectives are listed in Table I. Objective MinAvg means minimization of average number of steps E [T] needed for decoding and MaxPr maximizing the probability P3 of decoding in exactly three steps. Both uniform and degree-1 distributions perform rather poorly, the degree-1 distribution being worst. In contrast, the binomial distribution performs reasonably well, next followed by the soliton distribution (in fact, these distributions are similar). Also the ranking of the other four distributions is the same as before. Binomial and soliton distributions work reasonably well with respect to both criteria (though, in relative terms, not quite as well as in the case N = 3), being almost equal (again the distributions are in this case very similar), while the other two are much poorer.

- When the objective of optimization is maximization of the probability PN of success-

ful decoding after reception of N encoded packets, i.e. at the first instant complete decoding is possible, the problem can be analyzed using another approach. The central observation is that in this case the order in which the packets are received is irrelevant; whatever the order of the N packets the decoding either is or is not successful at the moment when all the N packets have been received. The probability of success can then be found by a recursive combinatorial approach as detailed below. A. Recursive algorithm For completeness, let us now write Pn = Pn(p1, . . . , pn). In this function we allow degree distributions with a positive probability for an empty packet and define implicitly p0 = 1 - ToBeInserted n i=1 pi. Obviously we have P0 = 1 and P1(p1) = p1, which provide the seeds for the recursion. In order to calculate Pn(p1, . . . , pn) we condition this probability on n-m of the n received packets having degree 1, which happens with a probability equal to the (n - m)th point probability of the binomial distribution Bin(n, p1). For successful decoding one must necessarily have n - m ToBeInserted 1, otherwise the decoding does not get started. Further, because the successful decoding after n received packets requires that no packets are wasted there must be no duplicates and all the n - m degree-1 packets must be distinct. This happens with the probability (n - 1)!/m! nn-m-1. Given the n - m distinct degree-1 packets, we have a remaining decoding problem for the m other packets that originally are surely at least of degree 2, but whose degrees may be modified when the n-m degree-1 packets are removed from the other packets in the decoding process, giving B. Numerical results For the cases N = 5, . . . , 8 the symbolic expressions of type (5) can be relatively easily handled and optimized numerically. The results are shown in Table III, where the estimates for E [T] were obtained by simulations. An example of the optimal degree distribution obtained using the numerical recursion for somewhat greater value N = 16 is shown in Fig. 2, with the optimum P16 = 0.01551. The distribution has a strikingly irregular character. Same kind of irregularities do also show up already for smaller values of N in Table III. It should, however, be noted that the results exhibit a great degree of insensitivity with respect to some features of the degree distribution. This is demonstrated by the fact that the same maximal value P16 = 0.01551 is obtained for instance with a distribution where only the probabilities p1 = 0.1565, p2 = 0.5493, p4 = 0.2095, p8 = 0.0732 and p16 = 0.0115 are non-zero. In order to better understand the nature of this kind of insensitivity we calculated the second derivative matrix Aij = Delta 2PN/Delta piDelta pj , i, j = 2, . . . , N at the optimum point with the variable p1 eliminated by the norm condition ToBeInserted i pi = 1. The eigenvectors and eigenvalues of A determine the principal directions and curvatures in these directions. It turns out that the eigenvalues constitute a rapidly decreasing sequence, with the ratio of two consecutive eigenvalues being of the order of 10. For instance in the case N = 3 (A is a 2 x 2 matrix) the eigenvalues are Lambda 1 = -6.97 and Lambda 2 = -0.71. The P3 surface is illustrated in Fig. 3, where also the principal directions are shown. The function forms a ridge which is steep in one direction but flat in the direction along the top of the ridge. The effect is much more pronounced when N is larger. For instance, for N=10 the largest and smallest eigenvalues are Lambda 1=-27.3 and Lambda 9=-2.53 x 10-6 showing that the function is extremely insensitive to changes in the direction of the last eigenvector. In fact, when the vector representing the distribution is constrained to lie on the intersection of hyperplanes with normals given by a few first eigenvectors and going through the optimal point, the PN has almost the optimal value no matter what the location of the distribution vector is in the other directions. Our experiments indicate that

typically three conditions of this type suffice to define a distribution which performs very close to the optimum. Fig. 4 shows the probability of successful decoding PN after N received packets as a function of N for N = 1, . . . , 20. The results have been obtained by using the optimized degree distribution for each N. Also shown is the behaviour of the relative overhead (E [T] - N)/N. This was obtained using a degree distributions optimized not for the MinAvg criterion but for the MaxPr criterion as above, and estimating E [T] by simulations. As is well known, for such small values of N the performance of LT codes is rather poor. The highest relative overhead 44.6 decline of the relative overhead starts but the codes become efficient only when N is of the order of 10000.

(Cri says: "This is why we then focus of the analysis of the performance for big values of N:")

LT Decoding Drawbacks In the LT decoding process, a symbol is taken off from the ripple each time and the related packets are updated accordingly. The process terminates when there is no more symbol left in the ripple, which is said to be successful if all symbols are recovered. Hence, in the process, it is important not to let the symbols in the ripple run out before all symbols can be recovered. Since the arrival of new symbols into the the degree distribution used in the encoding is also a crucial factor to the final success of the decoding process. Our further investigation on the LT decoding process reveals that when LT decoding process terminates and reports a failure, often the undecodable packets can be decoded to recover all symbols within the packets by other means, such as Gaussian elimination [10]. In other words, when viewing a packet as an equation formed by combining linearly a number of variables (or symbols) in GF(2), the set of available equations (or packets) may give a full rank such that a numerical solver (or decoder) can determine all variables (or symbols). Attributing to the design of the LT decoding process, the method recovers only partial but not all symbols. B. Full Rank Decoding Knowing that the LT decoding process terminates early due to lack of symbols in the ripple, in this paper, we propose a method, called full rank decoding to extend the decodability of LT codes. Specifically, given a set of packets of full rank, whenever the ripple is empty causing an early termination, a particular symbol is borrowed and decoded through some other method, and then placed into the ripple for the LT decoding process to continue. This procedure is repeated until the LT decoding process terminates with a success. Note that in the case of full rank, any picked borrowed symbol can be decoded with a suitable method to allow continuation of the LT decoding process. Unlike the previous approach [8], [10] where the decoding procedure completely or partially relies on Gaussian elimination, our proposed decoding algorithm mainly uses LT decoding to recover symbols, and triggers Wiedemann algorithm when LT decoding fails in order to recover a borrowed symbol, then returns back to LT decoding to recover subsequent symbols. We use the following criteria to decide which symbol should be borrowed. Considering decoding k symbols with n packets, given the above analogy between a packet and a linear equation, let v1; v2; : : : ; vn denote the coefficient vectors with the entries in GF(2) at the intermediate decoding process when the ripple is empty. Let bj denote the index of the borrowed symbol, and bj is decided by bj = arg maxj (V (j)jV := Pn i=1 vi) : (1) The above sum operation is defined in real number field, and V (j) gives the value at index j of the vector V. The basic idea here is to choose the symbol that is carried by most packets. Incorporating borrowed symbols into the LT decoding, our proposed full rank LT decoding is shown in Algorithm 1. C. Recovering the Borrowed Symbol One of the critical designs in our proposed idea is the recovery of a borrowed symbol. We need to seek for a suitable method that can recover only a single symbol using a low computational cost. Common techniques such as Gaussian elimination are inadequate as they are designed to recover all present our method in the following. Let M denote the coefficient matrix. M is of size

n£k for decoding k symbols with n packets. To ease our discussion, we let n = k for the time being. The case of n > k will be discussed later. The task of decoding is essentially to solve Mx = y (2) for x where x is the k£l vector of the symbols, y is the n£l vector of received packets, l is the length of a symbol, and M is defined over GF(2). Rather than the entire symbol set, our decoding procedure only needs to solve for a particular symbol, i.e. the borrowed symbol. This allows us to use a solver that deals with solving for only a single symbol with much lesser computational cost. The first task is to identify a collection of row vectors fromM such that the row vectors eliminates all other symbols except the borrowed symbol. Let x0 be a k £ 1 vector over GF(2) describes the selection of row vectors, where x0 j = 1 indicates that the jth row vector is selected and x0 j = 0 indicates otherwise. Let ui be a k £ 1 unit vector where the unique 1 locates at the index i which is also the index of the borrowed symbol. We then need to solve the following Mtx0 = ui (3) for x0 where Mt = MT. The fact that M is of full rank guarantees the existence of a solution. Finally, the inner product of (x0; y) gives the borrowed symbol. We use the efficient Wiedemann algorithm [11] to solve (3). The vector ui is first used to generate the so-called Krylov sequence ui;Mtui;M2t ui; :::;Mkt ui; ::: : Let S be the space spanned by this sequence, where Mt D. Non-square Case In practice, it is likely that more than k packets have to be received in order to ensure the complete recovery of all symbols. As a result, the coefficient matrix M will be nonsquare, i.e., the matrix is of size n £ k with rank equal to k and n ToBeInserted k IV. NUMERICAL RESULTS AND DISCUSSION Reference [6] and Section III provide the probabilities that all symbols can be successfully decoded after receiving a particular number of packets for LT and FR decoding algorithms respectively. Using these results, in Fig. 3, we compare the expected number of packets received for both algorithms. Four different set of configuration parameters are used, and the number of source symbols k ranges from 30 to 100. We compare with two sets of configuration parameters related to LT codes where both show consistent results. From Fig. 3, it can be seen that our proposed full ranking decoding yields much better performance in terms of the expected number of packets needed for decoding. It can also be seen that the performance of our proposed algorithm is close to the ideal performance (i.e. n = k). Conversely, LT codes need additional packets to compensate for the imperfectness of received degree distribution to avoid the decoding process from terminating prematurely. We further plot the benchmark the performance of both decoding schemes in Fig. 4. We develop both the decoding algorithms in Matlab and use tic/toc commands to record the runtime for decoding k symbols of 16 bits with 95probability. It can be seen our FR decoding requires much lower runtime for both configurations. While FR decoding requires additional steps to recover the borrowed symbols, it permits fewer input packets to decode which reduces the overall processing time. In addition, unlike traditional Wiedemann or Gaussian Elimination methods, the size of the coefficient matrix keeps on shrinking with each round of LT decoding passed.

3.3.2 Relay techniques

collaboration have been proposed to increase system performance [2]. In this work, diversity is introduced at the packet level. Instead of directly forwarding the received packet, the nodes store and re-transmit a linear combination of the previously received packets from their buffer. As a consequence, at the destination, the packets are likely to contain diverse combinations of the initial binary data: this is referred to as packet diversity in this paper. By this way, redundancy is introduced without the overload drawback of basic packet repetition. In [3], the authors presented this technique for Random Linear code (RL code). However, in a WSN with limited resources, the use of RL code is unsuitable due to its high decoding complexity. To avoid this problem, in this paper, we propose to focus on a more practical code called Luby Transform code (LT code). The idea proposed in [3], where relay nodes simply combine packets using XOR operations, is more attractive

for WSN. Although the LT code itself has been shown to provide several benefits for WSN [5], relaying schemes using an arbitrary XOR operation between LT encoded packets are subject to several constraints. Indeed, simply applying XOR operation as in [3] becomes inefficient because the global degree distribution of the information flow arriving at the destination is altered and leads to decoding inefficiency.

NOTE: the Results:

For any link within a linear network, we have considered an erasure channel model characterized by the same Packet Error Rate (PER). To highlight the positive impact of our proposed relaying strategies, we focus on a high PER. An extra header of size K is added to each packet. The indices corresponding to the fragments of the original packet that are contained in the encoded packet are set to 1 while others are set to 0. Consequently, the header of a XOR-ed packet results from a XOR combination among the headers of the original packets. We represent our results for two distinct cases when ML and BP decoding are used as shown in Fig. 2 and Fig. 3 respectively. Although LT code is meant to be decoded using BP algorithm, the results for the ML-decoding case are presented as well. Indeed, contrary to BP, ML-decoding is not biased by the shift in the degree distribution and thus provides an optimal bound with respect to the coding overhead. The hop-by-hop scenario gives the lowest bound on the number of transmissions required from the source to achieve transmission success. However, it comes at a price of too high latency and computational cost. Hence, we aim at approaching this optimum using alternative strategies with respect to WSNs requirements. Our results are evaluated in term of the overload factor which is defined as the ratio between the number of transmissions required from the source for each strategy and the number of transmissions required for the hop-by-hop scenario. Our objective is to minimize this overload factor. The results from Fig. 2 confirm that XOR-ing along the line reduces overload factor thanks to packet diversity introduced in the system. The more packets are combined together, the higher the robustness becomes. This behavior is however different with BP-decoding. For the XOR combination of 5 last packets scenario, the overload factor is at its best with ML, but for the BP-decoder, the severe alteration of the degree distribution limits its ability to decode. The degradation of degree distribution after 10 hops is shown in Fig. 4. Compared to the RSD, strategies 3, 5 and 6 provide the mean square error of the received degree distribution equal to 3.80 x 10-5 , 8.14x10-4 and 6.25 x 10-5 respectively. For the BP-decoder, the basic strategy where the Last Packet is retransmitted performs better than the retransmission of a XOR combination with prescribed degree. As we can see from Fig. 4, low degree packets are difficult to be obtained from Algorithm 1. In contrast, the LT-Adapted XOR combination with prescribed degree method preserves the Robust Soliton Distribution and leads to an efficient decoding process with a high performance improvement. The optimal XOR-ing probability (PXOR) for this specific configuration is approximately 0.2. With BP-decoding as shown in Fig. 3, apart from the hopby- hop Decode and Forward strategy that is mentioned to be unsuitable for WSN, our proposed relaying technique outperforms all other realistic existing relaying schemes because it represents a good tradeoff between an increased diversity and the decoding capability. **(Cri says: An application: BER for unequal error protection)**

Paper 14 **(Cri says: Nontheless it is meaningful to go into details for what concerns Raptor codes)**

Elias showed that the capacity of the BEC with erasure probability equals . He further proved that random codes of rates arbitrarily close to can be decoded on this channel with an exponentially small error probability using maximumlikelihood (ML) decoding. In the case of the erasure channel, ML decoding of linear codes is equivalent to solving systems of linear equations. This task can be done in polynomial time using Gaussian elimination. However, Gaussian elimination is not fast enough, especially when the length of the code is long. Reed–Solomon codes can be used to partially compensate for the inefficiency of random codes. Reed–Solomon codes can be de-

coded from a block with the maximum possible number of erasures in time quadratic in the dimension. (There are faster algorithms based on fast polynomial arithmetic, but these algorithms are often too complicated in practice.) However, quadratic running times are still too large for many applications. In [2], the authors construct codes with linear time encoding and decoding algorithms that can come arbitrarily close to the capacity of the BEC. These codes, called Tornado codes, are very similar to Gallager's low-density parity-check (LDPC) codes [3], but they use a highly irregular weight distribution for the underlying graphs.

schemes, a new class of codes is needed. Fountain codes constitute such a class, and they address all the above mentioned issues.

For many applications it is important to construct universal Fountain codes for which the average weight of an output symbol is a constant and which have fast decoding algorithms. In this paper, we introduce such a class of Fountain codes, called Raptor codes
The results of the previous section imply that LT-codes cannot be encoded with constant cost if the number of collected output symbols is close to the number of input symbols. tion 1. The decoding graph needs to have an order of edges in order to make sure that all the input nodes are covered with high probability. The idea of Raptor coding is to relax this condition and require that only a constant fraction of the input symbols be recoverable. Then the same information-theoretic argument as before shows only a linear lower bound for the number of edges in the decoding graph. There are two potential problems with this approach: 1) The information-theoretic lower bound may not be matchable with an algorithm, and 2) we need to recover all the input symbols, not only a constant fraction. The second issue is addressed easily: we encode the input symbols using a traditional erasure correcting code, and then apply an appropriate LT-code to the new set of symbols in a way that the traditional code is capable of recovering all the input symbols even in face of a fixed fraction of erasures. To deal with the first issue, we need to design the traditional code and the LT-code appropriately. Let be a linear code of block length and dimension , and let be a degree distribution.ARaptor code with parameters is an LT-code with distribution on symbols which are the coordinates of codewords in . The code is called the pre-code of the Raptor code. The input symbols of a Raptor code are the symbols used to construct the codeword in consisting of intermediate symbols.
D. Combined Error Probability The decoding error probability of a Raptor code with parameters can be estimated using the finite-length analysis of the corresponding LT-code and of the pre-code . This can be done for any code with a decoder for which the decoding error probability is completely known. For example, can be chosen from the ensemble . We will assume throughout that the input symbols of the Raptor code need to be recovered from output symbols. Suppose that has block length . For any with , let denote the probability that the LT-decoder fails after recovering of the intermediate symbols. Further, let denote the probability that the code cannot decode erasures at random positions. Since the LT-decoding process is independent of the choice of , the set of unrecovered intermediate symbols at the point of failure of the decoder is random. Therefore, if denotes the probability that the input symbols cannot be recovered from the output symbols, then we have Using the results of the previous two subsections, it is possible to obtain good upper bounds on the overall error probability of Raptor codes
obtain the value of the output symbol. Asimple probabilistic analysis shows that for maximum-likelihoos (ML) decoding to have a vanishing error probability for an LT-code, the average degree of an output symbol has to grow at least logarithmically with the number of input symbols. This makes it very difficult to obtain a linear time encoder and decoder for an LT-code. Raptor codes [16] are an extension of LT-codes which solve this problem and yield easy linear time encoders and decoders. The main idea behind Raptor codes is to pre-code the input symbols using a block code with a linear time encoder and decoder. The output symbols are produced using the original input symbols

together with the redundant symbols of the pre-code. Raptor codes solve the transmission problem over an unknown erasure channel in an almost optimal manner, as described in [16]. The success of Fountain codes for the erasure channel suggests that similar results may also be possible for other binarysymmetric channels. In this paper, we will investigate this question. As we will show, some of the properties of LT- and Raptor codes over the erasure channel can be carried over to any binary input memoryless symmetric channels (BIMSC), while some other properties cannot.
In this paper we will study BIMSCs. Three examples of such channels are furnished by the BEC with erasure probability , denoted BEC , the BSC with error probability , denoted BSC , and the BIAWGN channel with variance , denoted BIAWGN . We consider transmission with binary antipodal signaling. Strictly speaking, with this kind of signaling, we cannot speak of XORing bits. However, we will abuse notation slightly and denote the real product of the input values as the "XOR" of the bits. The output of a BIMSC with binary input can be identified with a pair , where , and is a real number between and . The value is interpreted as a guess of the input value before transmission over the channel, and can be interpreted as the probability that the guess is incorrect. The channel can be identified with the probability density function (pdf) of the error probability . For example, BEC is identified with the probability distribution ,

Finally, we give a formal definition of the reception overhead of a decoder: For each received bit, let be the probability that the bit was zero before transmission, and let , where is the number of collected output bits to which the decoding algorithm is to be applied. We say that the decoding algorithm has a reception overhead of if . In other words, the algorithm works with a number of output bits that is only away from the optimal number. **NOTE:** The results In this section, we will report on simulations we performed for degree distributions that were optimized for the BEC, as reported in [16]. Our results are analogous to those of Palanki and Yedidia [22]. Our experiments used the output distribution (14) We chose a Raptor code with parameters , where is a right-Poisson, left-regular LDPC code of rate , as described in [16]. The communication channel is BIAWGN , and the simulations were done for various values of the standard deviation . The results of these simulations are summarized in Fig. 2.
We performed our experiments in the following way: each time we ran enough experiments to see 200 bit errors, or 2000 decoding runs, whichever was first, and we also ran at least 200 decoding runs. Then we calculated the average fraction of bit errors at the end of the decoding process. During the decoding process, we ran the BP algorithm for at most 300 rounds. Since the degree distribution was optimized for the BEC, the smallest overhead to ensure a good error probability is expected to occur for the case ; this is also what the simulations suggest. Moreover, as is increased, the corresponding overhead needs to be increased as well. The graphs in Fig. 2 clearly show the advantage of Raptor codes over LT-codes.

3.3.3 A comparison with BER bounds on raptor codes

Theorem 1: Consider a Raptor code with parameters omega(x), mu(x), KL, K, Q, r and expanding coefficient etha. Its upper BER bound under ML decoding over a fast Rayleigh fading channel is Pb leq min1, Raptor U , where Raptor U is given by

Consider a Raptor code with parameters omega(x), mu(x), KL, K, Q, r and expanding coefficient etha. Its lower BER bound under ML decoding over a fast Rayleigh fading channel is Pb geq max0, Raptor L , where Raptor L is given by

(Cri says: simulazioni) In this section, we present numerical and simulation results for the LT-based DNC scheme and the proposed Raptor-based DNC scheme over quasi-static Rayleigh fading channels. Due to the time-varying channels of the wireless network, within each transmission round, a Raptor code is generated on-the-fly to match the instanta-

neous network topology. Thus, the BER performance of an ensemble of codes is analyzed.We consider the case of K = 100, KL = 50 and 98, which correspond to the pre-code rate r = 0.5 and 0.98, respectively. We evaluate the upper and lower bounds on the bit error probability under ML decoding by using the