A Digital Fountain Approach to Asynchronous Reliable Multicast

John W. Byers, Michael Luby and Michael Mitzenmacher

Abstract—

The proliferation of applications that must reliably distribute large, rich content to a vast number of autonomous receivers motivates the design of new multicast and broadcast protocols. We describe an ideal, fully scalable protocol for these applications that we call a digital fountain. A digital fountain allows any number of heterogeneous receivers to acquire content with optimal efficiency at times of their choosing. Moreover, no feedback channels are needed to ensure reliable delivery, even in the face of high loss rates.

We develop a protocol that closely approximates a digital fountain using two new classes of erasure codes that for large block sizes are orders of magnitude faster than standard erasure codes. We provide performance measurements that demonstrate the feasibility of our approach and discuss the design, implementation and performance of an experimental system.

Keywords— **Reliable multicast, content delivery, forward error correction, erasure codes, scalability.**

I. Introduction

A natural solution for companies that plan to efficiently disseminate large, rich content over the Internet to millions of concurrent receivers is multicast or broadcast transmission. These transmissions must be fully reliable, have low network overhead, support vast numbers of receivers with heterogeneous characteristics, and should be deployed with a minimum of server-side infrastructure investment. Activities that have such requirements include distribution of software, archived video, financial information, music and games. One method for content dissemination is to "push" content from a single source to multiple receivers, which can be achieved by reliable multicast. But many applications require more than just a reliable multicast protocol, since receivers will wish to access the data at times of their choosing, their access speeds will vary, and their access times will overlap with those of other receivers. Our general approach will accommodate both of these application styles.

While unicast protocols successfully use receiver initiated requests for retransmission of lost data to provide reliability, it is widely known that the multicast analogue of this solution is unscalable. For example, consider a server distributing a new software release to thousands of receivers. As receivers lose packets, their requests for retransmission can quickly overwhelm the server in a process known as feedback implosion. Even in the event that the server can handle the requests, the retransmitted packets are often of use only to a small subset of the receivers. More sophisticated solutions that address these limitations by using techniques such as local repair, polling, or the use of a hierarchy have been proposed [10], [15], [26], [21], [34], but these solutions as yet appear inadequate [24]. Moreover, whereas adaptive retransmission-based solutions are at best unscalable and inefficient on terrestrial networks, they are unworkable on satellite networks, where the back channel typically has high latency and limited capacity, if it is available at all.

The problems with solutions based on adaptive retransmission have led many researchers to consider applying Forward Error Correction based on erasure codes (also known as FEC codes) to reliable multicast [11], [22], [23], [25], [28], [29], [31], [32]. The basic principle behind the use of erasure codes is that the original source data, in the form of a sequence of k packets, along with additional redundant packets, are transmitted by the sender, and the redundant data can be used to recover lost source data at the receivers. A receiver can reconstruct the original source data once it receives a sufficient number of packets. The main benefit of this approach is that different receivers can recover from different lost packets using the same redundant data. In principle, this idea can greatly reduce the number of retransmissions, as a single retransmission of redundant data can potentially benefit many receivers simultaneously.

The work of Nonnenmacher, Biersack and Towsley [25] defines a hybrid approach to reliable multicast, coupling requests for retransmission with transmission of redundant codewords, and quantifies the benefits of this approach in practice. Their work, and the work of many other authors, focus on erasure codes based on Reed-Solomon codes [12], [22], [23], [28], [29], [31]. The limitation of these codes is that encoding and decoding times are slow on large block sizes (quadratic in the block size), effectively limiting k to small values for practical applications. Hence, their solution involves breaking the source data into small blocks of packets and encoding over these blocks. Receivers that have not received a packet from a given block request retransmission of an additional codeword from that block. They demonstrate that this approach is effective for dramatically reducing the number of retransmissions when

J. Byers is with the Dept. of Computer Science at Boston University and Digital Fountain, Inc. and is supported by National Science Foundation grant ANIR-9986397 and NSF CAREER award ANIR-0093296. E-mail: byers@cs.bu.edu. M. Luby is with Digital Fountain, Inc., Fremont, CA, E-mail: luby@digitalfountain.com. J. Byers and M. Luby initiated this research while with the International Computer Science Institute, Berkeley, California, with the support of NSF grant NCR-9416101. M. Mitzenmacher is with the Division of Engineering and Applied Sciences at Harvard University and is supported in part by National Science Foundation operating grants CCR-9983832, CCR-0118701, CCR-0121154, and by an Alfred P. Sloan Research Fellowship. E-mail: michaelm@eecs.harvard.edu. M. Mitzenmacher initiated this research while at Digital Systems Research Center, Palo Alto, California; part of the work was done while the author was visiting Digital Fountain, Inc. A preliminary version of this paper appeared in Proc. of ACM SIGCOMM '98 in Vancouver, Canada [8].

packet loss rates are low (they typically consider 1% loss rates). However, this approach cannot eliminate the need for retransmissions, especially as the number of receivers grows large or for higher rates of packet loss. This general approach also does not enable receivers to join the session dynamically.

To eliminate the need for retransmission and to allow receivers to access data asynchronously, the use of a *data carousel* or broadcast disk approach can ensure full reliability [1]. In a data carousel approach, the source repeatedly loops through transmission of all data packets. Receivers may join the stream at any time, then listen until they receive all distinct packets comprising the transmission. Clearly, the reception overhead at a receiver, measured in terms of unnecessary receptions, can be extremely high using this approach. As shown in [2], [29], [31], adding redundant codewords to the carousel can dramatically reduce reception overhead. These papers advocate adding a fixed amount of redundancy to blocks of the transmission using Reed-Solomon codes. The source then repeatedly loops through the set of blocks, transmitting one data or redundant packet about each block in turn until all packets are exhausted, and then repeats the process. This interleaved approach enables the receiver to reconstruct the source data once it receives sufficiently many packets from each block. The limitation of using this approach over lossy networks is that the receiver may still receive many unnecessary packets from blocks that have already been reconstructed while waiting for the last packets from the last few blocks it still needs to reconstruct. We quantify the performance cost of such an approach in Section VI.

The approaches described above that limit the need for retransmission requests can be thought of as imperfect approximations of an ideal solution, which we call a *digital fountain*. A digital fountain is conceptually simpler, more efficient, and applicable to a broader class of networks than previous approaches. A digital fountain injects a stream of distinct encoding packets into the network, from which a receiver can reconstruct the source data. The key property of a digital fountain is that the source data can be reconstructed intact from *any* subset of the encoding packets equal in total length to the source data. The digital fountain concept resembles ideas found in the seminal works of Maxemchuk [20] and Rabin [27]. Our approach is to construct better approximations of a digital fountain from fast erasure codes as a basis for protocols that perform reliable distribution of bulk data.

We emphasize that the digital fountain concept is quite general and can be applied in diverse network environments. For example, our framework for data distribution is applicable not only to multicast on the Internet, but also to satellite and wireless networks. These environments are quite different in terms of packet loss characteristics, congestion control mechanisms, and end-to-end latency; we strive to develop a solution independent of these environment-specific variables. These considerations motivate us to study, for example, a wide range of packet loss rates in our comparisons.

The body of the paper is organized as follows. In the next section, we describe in more detail the characteristics of the problems we consider. In Section III, we describe the digital fountain solution. In Section IV, we describe how to build a good theoretical approximation of a digital fountain using erasure codes. A major hurdle in implementing a digital fountain is that standard Reed-Solomon codes have unacceptably high running times for these applications. Hence, in Section V, we describe Tornado codes [18], a class of erasure codes that have extremely fast encoding and decoding algorithms, but which still do not realize all the benefits of a digital fountain solution. We then outline the properties of a new class of codes, LT codes [17], which for all practical purposes, realize the digital fountain solution, and have been used commercially in the products of Digital Fountain, Inc. [9]. Both of these classes of codes generally yield a far superior approximation to a digital fountain than can be realized with Reed-Solomon codes in practice, as we show in Section VI. In Section VII, we describe the design and performance of a working prototype system for bulk data distribution based on Tornado codes that is built on top of IP Multicast. The performance of the prototype bears out the simulation results, and it also demonstrates the interoperability of this work with the layered multicast techniques of [32], [6] and others. We conclude with additional research directions we are pursuing which instantiate the digital fountain approach for other content distribution methods.

II. Requirements for an Ideal Protocol

We recall an example application in which millions of receivers want to download a new release of software over the course of several days. For this application, we assume that there is a single distribution server, and that the server will send out a stream of packets (using either broadcast or multicast) as long as there are receivers 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 protocol for distributing the software using multicast should be:

- **Scalable:** Server load remains constant whether there are one or a million receivers.
- **Reliable:** An exact copy of the original file is reconstructed by each receiver.
- **Reception-efficient:** The total number of packets each receiver needs to reconstruct the file is minimal. Ideally, the aggregate length of packets needed is equal to the length of the original file.
- **Time-efficient:** The amount of processing required to generate packets at the server and to reconstruct the file from received packets at the receiver is minimal.
- **Time-independent:** Receivers may initiate the download at their discretion, implying that different receivers may start the download at widely disparate times. Receivers may sporadically be interrupted and continue the download at a later time.

• **Server-independent:** Receivers may collect packets for the file from one or more servers that are transmitting packets. No coordination between servers should be required for this.

• **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 wired Internet, or group communication over satellite, wireless, and cable are representative of channels we consider. Perhaps the most important property of these channels is that the return feedback channel from the receivers to the server is typically of limited capacity, or is non-existent. This is especially applicable to satellite transmission. These channels are generally packet based, and each packet has a header including a unique identifier. They are best-effort channels designed to attempt to deliver all packets, but frequently packets are lost or corrupted. Wireless networks are particularly prone to high rates of packet loss and all of the networks we describe are prone to bursty loss periods. We assume that error-correcting codes (used independently from the erasure codes we consider) are used to detect and correct errors within a packet. If a packet contains more errors than can be corrected by these codes, the error detection is used to recognize that the packet still contains uncorrectable errors and the packet is discarded and treated as a loss. Thus a packet either arrives completely intact and error-free, or it is lost.

The requirement that the solution be reliable, reception-efficient, and either time-independent, server-independent or tolerant implies that receiver robustness to any pattern of missing packets is crucial. For example, a receiver may sporadically be interrupted, resuming the download one or more times before completion. During the interruptions the server will continue sending out a stream of packets (to other interested receivers) that an interrupted receiver will miss. The efficiency requirement implies that the total length of all the content that a receiver must receive in order to recover the file should be approximately equal to the total length of the file.

III. The Digital Fountain Solution

In this section, we outline an idealized solution that achieves all the objectives laid out in the previous section for the channels of interest to us. In subsequent sections, we describe and evaluate a new approach that implements an approximation to this ideal solution that is superior to previous approaches.

A universe of receivers wish to acquire a source file. In the idealized solution, one or more servers send out a stream of encoding packets (each packet distinct from all others) that contain encoding data generated from the source file. The servers will generate and transmit encoding packets whenever there are any receivers joined to the sessions carrying the packets. A client remains joined to sessions from a subset of the servers until the aggregate length of all encoding packets it has received is equal to the length of the source file. In this idealized solution, each receiver can reconstruct an exact copy of the original file from the received encoding packets, independent of which servers generated the encoding packets, independent of losses, and independent of the intervals of time the receiver was joined to the sessions. Ideally, the amount of processing required by the servers to generate encoding packets and by the receivers to reconstruct the file from received encoding packets is minimal.

We metaphorically describe the stream of encoding packets produced by one of the servers in this idealized solution as a *digital fountain*. The digital fountain has properties similar to a fountain of water; drinking a glass of water, irrespective of the particular drops that fill the glass, quenches one's thirst. The digital fountain approach has all the desirable properties listed in the previous section and functions over channels with the characteristics outlined in the previous section.

An ideal way to implement a digital fountain is to directly use an erasure code that takes source data consisting of k source 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 encoded packets. But as we shall show in the next section, a straightforward implementation of a digital fountain using a Reed-Solomon code is impractical.

IV. Limitations of Building a Digital Fountain with Reed-Solomon Codes

We now consider implementation issues associated with building a digital fountain from Reed-Solomon codes. There are two related considerations. The first is running time, specifically the time it takes to generate an encoding, and the time it takes to decode. The second and more subtle consideration is a practical limitation on the size of an encoding that can be generated.

We begin with some terminology. Erasure codes are typically used to stretch a file consisting of k source packets into n encoding packets, where both k and n are input parameters. We refer to $s = n/k$ as the *stretch factor* of an erasure code. This finite stretch factor naturally limits the extent to which erasure codes can approximate a digital fountain; a reasonable approximation proposed by other researchers (e.g., [23], [28], [29], [32]), is to set n to be a multiple of k, then repeatedly cycle through transmission of these n encoding packets.

For Reed-Solomon codes, the size of the finite field symbol alphabet is an upper bound on n, and this size limits the stretch factor. In most practical implementations, the alphabet size is 256 (each symbol is one byte), which limits n to values of 256 or less. It is possible to use a larger alphabet size for Reed-Solomon codes, e.g., 65,536 (each symbol is two bytes), but in this case the practical stretch factor is severely limited to small values due to processing considerations. On the encoding side, the operations needed to generate n encoding packets requires $k(n-k)A/2$

exclusive-ors of source packets, where A is the length of a symbol (16 in this example). Thus, for example, if $k =$ 10,000, and $n =$ 20,000 (a moderate stretch factor of 2), then it takes 800,000,000 exclusive-ors of source packets to produce 20,000 encoding packets from 10,000 source packets, or around 80,000 exclusive-ors of source packets per source packet, which is prohibitively expensive. Thus even a moderate stretch factor of 2 is not practically possible for moderate values of k. For all but very small values of k the practical limitation on $n - k$ is a few hundred at most, as the processing overhead to produce the encoding per source packet is linear in $(n - k)A/2$. (Values used in [25], [29], [32], [31] have k and $n-k$ ranging from 8 to 256.) The decoding time is typically comparable to the encoding time for Reed-Solomon codes.

There are several other significant limitations with constant stretch factors. The first limitation regards packet loss in the network — for any pre-specified value of s, under sufficiently high loss rates a receiver may not receive k out of n packets in one cycle. In such a setting, a receiver may receive useless duplicate transmissions in cycles subsequent to the first before being able to reconstruct the source data, thereby decreasing the channel efficiency. A similar effect would occur if a user wished to pause during a download. Upon resuming a download, the receiver might obtain a significant number of duplicate transmissions depending on where in the cycle they resume. A more significant limitation arises when receivers do not obtain transmitted packets for reasons other than packet loss in the network. An example we consider in more detail in Section VII-B is that of a cumulative layered multicast scheme used to serve heterogeneous receivers with different transfer rates. Slow receivers receive only a fraction of the packets transmitted to the fastest receiver by virtue of their lower subscription level. Hence data encoded with a constant stretch factor must be scheduled very carefully among the layers to reduce the incidence of duplicate transmissions. Similar considerations arise in the case of a parallel download, where a source may download encoding packets from several receivers concurrently. A finite stretch factor requires proper scheduling to reduce the incidence of duplicate transmissions, as explained in [7]. For all of these reasons, systems where encoding packets can be generated on-the-fly may be preferable in many situations.

The first alternative we propose to avoid the problems of Reed-Solomon codes is to use Tornado codes [18]. The main drawback of using Tornado codes relative to Reed-Solomon 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 VI. We also suggest a second alternative, LT codes [17]. Whereas Tornado codes admit only moderate stretch factors and have a decoding time that depends on n, LT codes can generate encodings with effectively unbounded stretch factors *and* can be decoded in time depending only on k (and not n). We provide a description of the constructions used to build these codes and their properties in the next section.

V. Tornado and LT Codes

In this section, we describe in some detail the construction of a specific Tornado code and explain some of the general principles behind Tornado codes. We also briefly describe the properties of the LT code, which are similar in some respects to Tornado codes, but exhibit additional key properties which provide a better approximation to an idealized digital fountain. We first outline how the theoretical basis for these codes differs from the traditional Reed-Solomon (RS) erasure codes. Then we give a specific example of a Tornado code based on [18], [19] and compare its performance to a standard RS 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 ℓ redundant packets for a total of $n = k + \ell$ encoding packets all of a fixed length P.

A. 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 x_i, and the jth redundant packet as containing the value of a variable y_j that is a linear combination of $\{x_i\}_{i=1}^{k}$ 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 $y_3 = x_1 + x_2\alpha + \ldots + x_k\alpha^{k-1}$, where α 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.

RS codes guarantee that successful receipt of any k distinct packets enables reconstruction of the source data. When e redundant packets and $k - e$ 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 $(k-e+1)e$. The remaining subsystem has e equations and e unknowns corresponding to the source data packets not received. With RS codes, this system has a special form that allows one to solve for the unknowns in time proportional to e^2 via a matrix inversion and matrix multiplication. Theoretical work demonstrates methods for Reed-Solomon encoding and decoding which are asymptotically faster than quadratic time, but nevertheless perform more slowly than the quadratic algorithms for practical values of k and l.

The large decoding time for RS codes arises from the *dense* system of linear equations used. Both Tornado and LT codes are built using a set of 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 paid 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. Designing the proper structure for

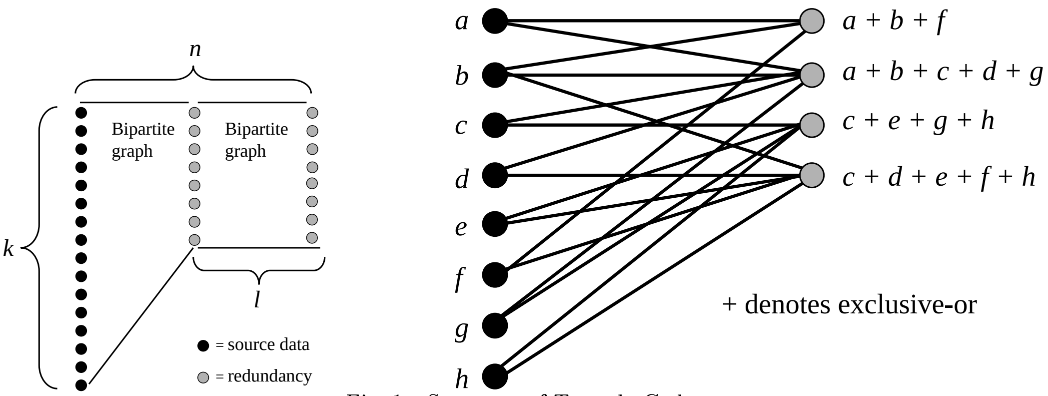

Fig. 1. Structure of Tornado Codes

the system of equations so that the number of additional packets and the coding times are simultaneously small is a difficult challenge [18], [19].

For both Tornado and LT codes, the linear equations have the form $y_3 = x_1 \oplus x_4 \oplus x_7$, where $\oplus$ is bitwise exclusive-or. Tornado codes also use equations of the form $y_{53} = y_3 \oplus y_7 \oplus y_{13}$; that is, redundant packets may be derived from other redundant packets, and in general there may be several layers of redundant packets each depending on the previous layer of packets. For Tornado codes, the equations of various forms are carefully chosen in advance. In particular, the number of encoding packets n must be predetermined before encoding, and thus the stretch factor for Tornado codes is fixed at encoding time. For practical purposes, the stretch factor for Tornado codes is restricted to be a small multiple of k, i.e. the stretch factor is generally ten or less. This restriction causes similar limitations to those described earlier for Reed-Solomon codes, albeit not as severe. The number of exclusive-ors of source packets per source packet to produce the encoding is a small constant, e.g., in the Tornado Z implementation in Section 5.2 where $k = 16,000$ and $n = 32,000$, this constant is 14. Thus, unlike Reed-Solomon codes, the encoding time per source packet does not grow as k and n grow. The decoding time for Tornado codes is essentially the same as the encoding time.

For LT codes, there is no predetermined value of n, as the equations placed into each encoding packet are generated independently of all other encoding packets. Thus, the stretch factor for LT codes is inherently unlimited, as an unlimited number of encoding packets can be generated. For LT codes the average number of exclusive-ors to generate each encoding packet and to reconstruct each source packet is upper bounded by the average number of variables in each equation. For values of k that are in the hundreds of thousands, for current commercial implementations of LT codes, this average is around 20. Thus, unlike Reed-Solomon codes, the average time to produce each encoding packet and to decode each source packet does not grow dramatically as k grows.

The decoding process for both Tornado and LT codes repeatedly uses the following simple *recovery rule*: Find any equation with exactly one variable, recover the value of the variable by setting it equal to the value of the equation (this uses up the equation), and then remove the newly recovered variable from any other equations that it appears in by exclusive-oring its value into each of these equations. For example, consider the equations $y_1 = x_3, y_2 = x_2 \oplus x_3, y_3 = x_3 \oplus x_1$ and $y_4 = x_4 \oplus x_2 \oplus x_1$. Then, applying the recovery rule repeatedly:

1. The value of x_3 is recovered from the equation $y_1 = x_3$. Then, the value of x_3 is XORed into y_2 and y_3 yielding new simplified equations $y_2' = x_2$ and $y_3' = x_1$, together with the unchanged equation $y_4 = x_4 \oplus x_2 \oplus x_1$.
2. The value of x_2 is recovered from the equation $y_2' = x_2$. Then, the value of x_2 is XORed into y_4 yielding the new simplified equation $y_4' = x_4 \oplus x_1$ together with the unchanged equations $y_3' = x_1$.
3. The value of x_1 is recovered from the equation $y_3' = x_1$. Then, the value of x_1 is XORed into y_4' yielding the new simplified equation $y_4'' = x_4$.
4. The value of x_4 is recovered from the equation $y_4'' = x_4$.

The applicability of the recovery rule usually remains minimal until slightly more than k encoding packets have arrived. Then, often the single arrival of a new encoding packet containing an equation triggers a whirlwind of applications of the rule, leading to the recovery of all k source packets. This whirlwind explains the origin of the name Tornado codes.

The decoding may stop as soon as enough packets arrive so that the source data can be reconstructed. Note that the fast erasure codes use only exclusive-or operations and avoid both the field operations and the matrix inversion inherent in decoding RS codes. The total number of exclusive-or operations for decoding is at most the number used for encoding, and in general is less.

For Tornado codes and LT codes, we say that the *decoding inefficiency* is $1 + \epsilon$ if $(1 + \epsilon)k$ encoding packets are required to reconstruct the source file consisting of k source packets. For both of these codes, the loss pattern associated with encoding packets is immaterial as to whether or not a receiver can recover the source file from a given number of encoding packets, but there is some variance in the number of encoding packets needed to recover the source file due to the randomness used by the encoding algorithms. For the Tornado Z code implementation described in Section 5.2, the decoding inefficiency is more than 1.06 with probability 1/10, and was not more than 1.10 in 10,000 trials. Nevertheless, these implementations of Tornado codes do not have tight enough bounds on the decoding inefficiency for commercial applications. For the current Digital Fountain commercial implementations of LT codes, the decoding inefficiency is more than 1.05 with probability less than 10^{-8} for almost any size source file.

One of the advantages of Tornado codes and LT codes over standard codes is that they trade-off a small degradation in decoding inefficiency for a substantial improvement in encoding and decoding times. Recall that Reed-Solomon codes have encoding and decoding times proportional to $k(n-k)AP$, where A is the size of the finite field symbol alphabet and P is the packet size. In contrast, Tornado codes have asymptotic in k encoding and decoding times proportional to $n\ln(1/\epsilon)P$ with decoding inefficiency $1+\epsilon$. A summary comparing the asymptotic properties of encoding packet generation and file reconstruction for Tornado codes and Reed-Solomon codes is provided in Table I.

Tornado codes have the drawback that a stretch factor must be predetermined before encoding takes place and that the decoding time is proportional to the stretched encoding length. Furthermore, the stretch factor in practice can only be a small multiple, e.g., 4. Tornado codes are not entirely suitable for situations where there are substantial loss rates, or when the receiver may request to receive only a fraction of the encoding packets transmitted, or when the receiver may receive encoding packets from multiple senders for the same file. LT codes do not share any of these limitations. With LT codes, each encoding packet is produced on-the-fly from an extremely large set of possibilities at the same average processing cost as every other encoding packet. Because of this, the fraction of duplicate encoding packets produced is tiny. Thus a receiver is unlikely to receive any significant number of duplicate encoding packets, even if receiving packets from multiple senders for the same file, and even if receiving only a small fraction of the generated encoding packets, independent of the loss pattern. The analysis of the decoding inefficiency for LT codes accounts for this possibility, and as stated previously the probability of failing to recover the source file from any set of $1.05k$ encoding packets is tiny for commercial implementations of LT codes. LT codes have asymptotic in k time per encoding packet of $\ln(k)P$ and decoding time for the source file of $k\ln(k)P$ with decoding inefficiency that is asymptotically 1. Specifically, a failure probability Δ for an LT code can be achieved with a decoding inefficiency of $1+O(\ln^2(k/\Delta)/\sqrt{k})$; note that this is less than $1+\epsilon$ for any constant ϵ for sufficiently large k. Thus, LT codes are a practical realization of an idealized digital fountain. A summary comparing the asymptotic properties of encoding packet generation and file reconstruction for LT codes and Reed-Solomon codes is provided in Table 2.

In the next section, we present an example of a fast Tornado code with decoding inefficiency $1+\epsilon \approx 1.054$ whose performance we compare directly with RS codes.

	Tornado	Reed-Solomon
Decoding inefficiency	$1+\epsilon$	1
Encoding times	$n\ln(1/\epsilon)P$	$k(n-k)AP$
Decoding times	$n\ln(1/\epsilon)P$	$k(n-k)AP$

TABLE I
PROPERTIES OF TORNADO VS. REED-SOLOMON CODES WHEN A FIXED STRETCH FACTOR IS EMPLOYED

	LT	Reed-Solomon
Decoding inefficiency	$1+O(\ln^2(k/\Delta)/\sqrt{k})$	1
Per packet encoding times	$\ln(k)P$	kAP
Decoding times	$k\ln(k)P$	$k(n-k)AP$

TABLE II
PROPERTIES OF LT VS. REED-SOLOMON ON-THE-FLY CODES.

B. An Example

We now provide a specific example of a Tornado code. It is convenient to describe the association between the variables and the equations in terms of a layered graph, as depicted in Figure 1. The nodes of the leftmost layer of the graph correspond to the source data. Subsequent layers contain the redundant data. Each redundant packet is the exclusive-or of the packets held in the neighboring nodes in the layer to the left, as depicted on the right side of Figure 1. The number of exclusive-or operations required for both encoding and decoding is thus dominated by the number of edges in the entire graph.

We specify the code by specifying the random graphs to place between consecutive layers. The mathematics behind this code, which we call Tornado Z, is described in [18], [19] and will not be covered here. This code has 16,000 source data nodes and 16,000 redundant nodes, i.e. it employs a stretch factor of two. The code uses three layers; the number of nodes in the layers are 16,000, 8,000 and 8,000 respectively.

The graph between the first two layers is the union of two subgraphs, G_1 and G_2. The graph G_1 is based on a *truncated heavy tail* distribution. We say that a layer has a truncated heavy tail distribution with parameter D when the fraction of nodes of degree i is $\frac{D+1}{Di(i-1)}$ for $i=2,\ldots,D+1$. The graph G_1 connects the 16,000 source data nodes to 7,840 of the nodes at the second layer (the remaining 160 nodes at the second layer are used in G_2). The node degrees on the left hand side are determined by the truncated heavy tail distribution, with $D=200$. For example, this means that there are $\frac{(16{,}000)(201)}{(200)(2)(1)} = 8{,}040$ nodes of degree 2 on the left hand side. Each edge is attached to a node chosen uniformly at random from the 7,840 on the right hand side.[1] The distribution of node degrees on the right

[1]Notice that this may yield some nodes of degree 0 on the right hand side; however, this happens with small probability, and such nodes can be removed. Also, there may be multiple edges between pairs of nodes. This does not affect the behavior of the algorithm dramatically, although the redistribution of such multiple edges improves

hand side is therefore Poisson.

In the second graph G_2, each of the 16,000 nodes on the left has degree 2. The nodes on the right are the remaining 160 nodes at the second layer, and each of these nodes has degree 200. The edges of G_2 are generated by randomly permuting the 32,000 edge slots on the left and connecting them in that permuted order to the 160 nodes on the right. The graph G_2 helps prevent small cycles in G_1 from halting progress during decoding.

The graph between the second and third layers of nodes uses a specific distribution, designed using a linear programming tool discussed in [18], [19]. The linear program is used to find graphs that have low decoding inefficiency. In this graph, all of the 8,000 nodes on the left have degree 12. On the right hand side there are 4,093 nodes of degree 5; 3,097 nodes of degree 6; 122 nodes of degree 33; 472 nodes of degree 34; 1 node of degree 141; 27 nodes of degree 170; and 188 nodes of degree 171. The connections between the edge slots on the left and right are selected by permuting the edge slots on the left randomly and then connecting them to the edge slots on the right. In total there are 222,516 edges in this graph, or approximately 14 edges per source data node. The sparseness of this graph enables the very fast encoding and decoding.

C. Performance

In practice, Tornado codes where values of k and ℓ are on the order of tens of thousands can be encoded and decoded in just a few seconds. In this section, we compare the efficiency of Tornado codes with standard codes that have been previously proposed for network applications [11], [25], [28], [29], [31], [32]. The erasure code listed in Tables III and IV as *Cauchy* [4] (available at [13]), is a standard implementation of Reed-Solomon erasure codes based on Cauchy matrices. The Tornado Z codes were designed as described earlier in this section. The implementations were not carefully optimized, so their running times could be improved by constant factors. All experiments were benchmarked on a Sun 167 MHz UltraSPARC 1 with 64 megabytes of RAM running Solaris 2.5.1. Although this hardware is no longer state of the art, the running times nevertheless reflect the essential asymptotic and behavioral difference between RS codes and Tornado codes. All runs are with packet length $P = 1$KB. For all runs, a file consisting of k packets is encoded into $n = 2k$ packets, i.e., the stretch factor is 2.

For the decoding of the Cauchy codes, we assume that $k/2$ original file packets and $k/2$ redundant packets were used to recover the original file. This assumption holds approximately when a carousel encoding with stretch factor 2 is used, so that roughly half the packets received are redundant packets.

Tornado Z has an average decoding inefficiency of 1.054, so on average $1.054 \cdot k/2$ original file packets and $1.054 \cdot k/2$ redundant packets were used to recover the original file. Our results demonstrate that Tornado codes can be

performance marginally.

Encoding Benchmarks		
	Reed-Solomon Codes	Tornado Codes
SIZE	Cauchy	Tornado Z
250 KB	4.6 seconds	0.11 seconds
500 KB	19 seconds	0.18 seconds
1 MB	93 seconds	0.29 seconds
2 MB	442 seconds	0.57 seconds
4 MB	1717 seconds	1.01 seconds
8 MB	6994 seconds	1.99 seconds
16M Bytes	30802 seconds	3.93 seconds

TABLE III
COMPARISON OF ENCODING TIMES.

Decoding Benchmarks		
	Reed-Solomon Codes	Tornado Codes
SIZE	Cauchy	Tornado Z
250 KB	2.06 seconds	0.18 seconds
500 KB	8.4 seconds	0.24 seconds
1 MB	40.5 seconds	0.31 seconds
2 MB	199 seconds	0.44 seconds
4 MB	800 seconds	0.74 seconds
8 MB	3166 seconds	1.28 seconds
16 MB	13629 seconds	2.27 seconds

TABLE IV
COMPARISON OF DECODING TIMES.

encoded and decoded much faster than RS codes, even for relatively small files.

We note that there is a small variation in the decoding inefficiency for decoding Tornado codes depending on which particular set of encoding packets are received. To study this variation, we ran 10,000 trials using the Tornado Z code. In Figure 2, we show the percentage of trials for which the receiver could not reconstruct the source data for specific values of the decoding inefficiency. For example, using Tornado Z codes with each node representing one packet, a decoding inefficiency of 1.064 corresponds to receiving $17,024 = 1.064 \cdot 16,000$ packets. Over 90% of the receivers were able to reconstruct the source data before receiving this many packets.

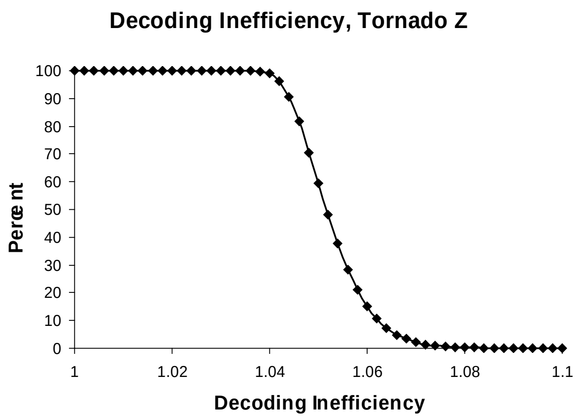

Fig. 2. Decoding inefficiency variation over 10,000 trials of Tornado Z.

In our trials the average decoding inefficiency was 1.0536, the maximum reception inefficiency was 1.10, and the standard deviation was 0.0073. For all 10,000 trials the same graph was used; this graph was *not* specially chosen, but was generated randomly as described in Section V-B. In practice one can achieve slightly better performance by testing various random graphs for performance before choosing one. Our tests suggest that the performance given in Figure 2 is representative.

VI. Simulation Comparisons

From the previous section, it is clear that using RS 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 [25], [28], [29], [31]. Interleaved codes are constructed as follows: suppose $K + L$ encoding packets are to be produced from K source packets. Partition the K source packets into blocks of length k, so that there are $B = K/k$ blocks in total. Stretch each block of k source packets to an encoding block of $k + \ell$ packets using a standard RS erasure code by adding $\ell = kL/K$ 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 encoding packets, each of which consist of exactly one packet from each block.

The choice of the value of the parameter k for interleaved RS 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 viewpoint of a receiver when at least k distinct encoding packets associated with that block have been received. The entire file can only be decoded by the receiver when all blocks are full. 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 encoding 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 $1 + \epsilon$ if one must obtain $(1 + \epsilon)k$ distinct encoding packets in order to decode the source data. For both Tornado and LT codes, there is some overhead in the decoding inefficiency due to the sparse nature of the codes and the randomness used in their construction. For interleaved codes, decoding inefficiency arises because in practice one must obtain more than k encoding 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.

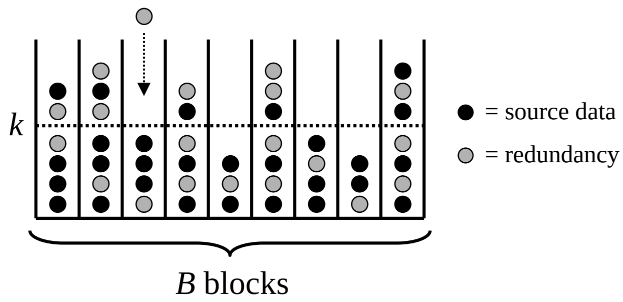

Fig. 3. Waiting for the last blocks to fill...

- 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?

While we have chosen Tornado Z codes to perform this comparison, a similar comparison can be made with the LT codes which have comparable decoding inefficiency and strictly faster decoding time than Tornado Z.

In our initial simulations, we assume probabilistic loss patterns in which each transmission to each receiver is lost independently with a fixed probability p. Using bursty loss models instead of this uniform loss model does 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 encoding packets in a random order from the source. Therefore, any loss pattern appears equivalent to a uniform loss pattern on the receiver end. The choice of the uniform loss 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 expect interleaved codes to have slightly *better* performance under random losses than under bursty losses. To confirm this intuition, in our previous work [8], we provided results from trace-driven simulations of Internet traffic taken from [33]. The following results focus on the random loss model, since this is the easiest model to analyze, it provides a lower bound on the decoding inefficiency for interleaved Reed-Solomon codes, and because in the cases we measured on the Internet trace data the difference in the decoding inefficiency for interleaved Reed-Solomon codes between random and bursty losses is small.

A. Equating Decoding Inefficiency

Our first simulation compares the decoding time of Tornado Z with an interleaved code with decoding inefficiency comparable to that of Tornado Z. In Section V, we deter-

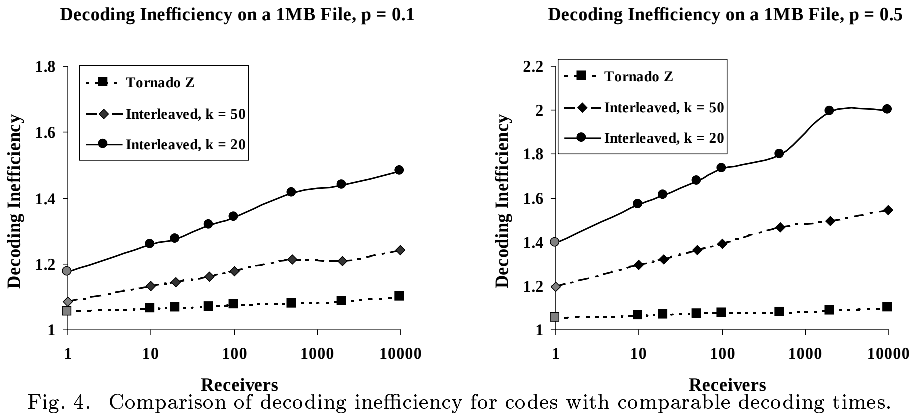

Fig. 4. Comparison of decoding inefficiency for codes with comparable decoding times.

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 V
SPEEDUP OF TORNADO Z CODES OVER INTERLEAVED RS CODES WITH COMPARABLE INEFFICIENCY.

mined experimentally that Tornado Z codes have the property that the decoding inefficiency is greater than 1.076 less than 1% of the time. In Table V, we present 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 V are derived. To compute the running time for interleaved codes, we 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% of the time. (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 per block decoding time, and multiply it 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 IV, we approximate the decoding time for a block of k source data packets by $k^2/31250$ seconds. To compute the running time for Tornado Z, we simply use the decode times for Tornado Z as given earlier in Table IV.

As an example, suppose the encoding of a 16 MB file is transmitted over a 1 Mbit/second channel with a loss rate of 50%. It takes just over 4 minutes to receive enough packets to decode the file using either Tornado Z or an interleaved code (with the desired decoding inefficiency guarantee). However, the decoding time is almost 8 minutes for the interleaved code compared with just over 2 seconds for Tornado Z. Comparisons of 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.

B. Equating Decoding Time

Our second set of simulations examines interleaved codes 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 10% and 50%. The 10% loss rates are representative of congested Internet connections, while the 50% loss rates are near the upper limits of what a mobile receiver 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 1% loss rates studied in [25], interleaved codes and Tornado have generally comparable performance.

Figure 4 shows, for different numbers of receivers, the worst case decoding inefficiency 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 graph corresponds to 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 10% and a block size of $k = 50$, 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.

C. 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 encoding packets a receiver 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" [16].) In contrast, the number of encoding packets the receivers require to reconstruct the source data using the 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 approach. This effect is completely avoided by using Tornado codes.

VII. Implementation of an Asynchronous Reliable Multicast Protocol

In this section, we describe our simulations for distributing bulk data to a large number of heterogeneous receivers who may access the data asynchronously. Our implementation is designed for the Internet using a protocol built on top of IP Multicast. We outline our techniques to handle receiver heterogeneity using layered multicast [21], [23] and describe how our digital fountain approach for reliability cleanly integrates with TCP-friendly, receiver-driven congestion control methods such as the work of Vicisano, Rizzo and Crowcroft [32]. In a companion paper [6], we consider multirate multicast congestion control (especially suitable for content encoded using a digital fountain approach) in its own right. While we note that the system developed and described here constitutes a feasibility study, we emphasize that we have leveraged this prototype design to create a completely functional multicast protocol which is now shipping in product form.

We expect that the digital fountain approach via fast erasure codes will also prove useful in other environments besides a multicast-enabled Internet, such as satellite or wireless based systems. In these settings, different channel characteristics would suggest different approaches for congestion control and tolerance to receiver heterogeneity. However, the general approach for reliability which we advocate would remain essentially the same, even under varying end-to-end bandwidths and packet loss rates.

We now present the design of our multicast protocol. The two main issues are the use of layered multicast and the approach the receiver uses to decode the message. Then we describe the experimental setup and performance results of our system.

A. Integration with Layered Congestion Control

The approach to accommodating receiver heterogeneity with appropriate congestion control mechanisms which we propose follows the lead of other authors who advocate *layered* multicast [21], [23], [32]. The main idea underlying this approach is to enable the source to transmit data across multiple multicast groups, thereby allowing the receivers to subscribe to an appropriate subset of these layers. When such a scheme is also used to provide congestion control, it is typically called a *multirate* congestion control scheme. Of course, practical considerations warrant keeping the number of multicast groups associated with a given source to a minimum. A receiver's subscription level is based on factors such as the bottleneck bandwidth en route to the source and network congestion. Basic ideas common to the proposed layered schemes are:

- The server transmits data over multiple layers, where the layers are ordered by increasing transmission rate.
- The layers are *cumulative*, i.e. a receiver subscribing to layer i also subscribes to all layers less than i. We say that a receiver subscribes to *level* i when it subscribes to layers 0 through i.

For example, in the simplest version of our implementation, we use geometrically increasing transmission rates: $B_i = 2^{i-1}$ is the rate of the ith layer. Thus, a receiver at subscription level i would receive bandwidth proportional to $2B_i$, for $i \geq 1$. A protocol with which our approach is compatible, and which we use in our basic evaluation is the scheme described in work of Vicisano, Rizzo and Crowcroft [32] that proposes the following novel ideas, summarized here briefly:

- Congestion control is achieved by the use of *synchronization points* (SP's) that are specially marked packets in the stream. A receiver can attempt to join a higher layer only immediately after an SP, and keeps track of the history of events only from the last SP. The rate at which SP's are sent in a layer is inversely proportional to the layer bandwidth, thus lower bandwidth receivers are given more frequent opportunities to add higher layers.
- Instead of explicit join attempts by receivers, the server generates periodic *bursts* during which packets are sent at twice the normal rate on each layer. This has the effect of creating network congestion conditions similar to those that receivers would experience following an explicit join. If a receiver witnesses no packet losses during and after the burst, it adds a layer at the next SP.
- Receivers also use packet loss events as an indication of congestion. If a receiver witnesses packet losses during or after a burst, it does not add a layer at the next SP; moreover, receivers drop to a *lower* subscription level whenever

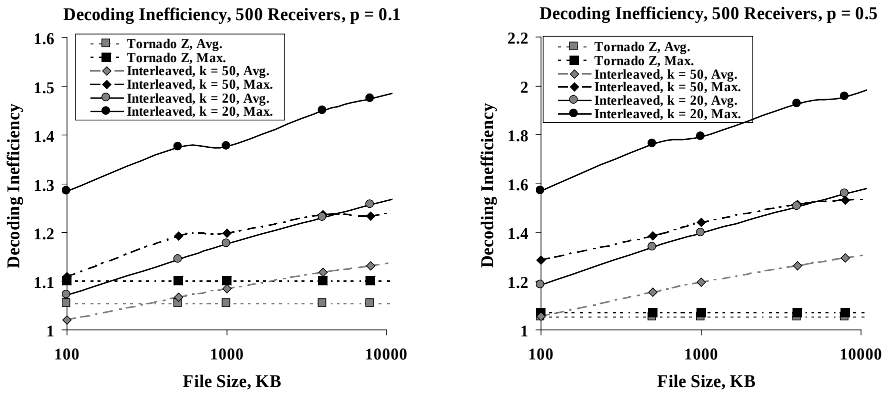

Fig. 5. Comparison of decoding inefficiency as file size grows.

a packet loss event occurs outside of a burst

Both the sending of SP's and burst periods are driven by the *sender*, with the receivers reacting appropriately. The most attractive feature of this approach is scalability — since the transmission schedule at the sender is fixed in advance, the sender behaves the same whether there are a handful or a million receivers. Moreover, this method of congestion control prevents feedback implosion as it does not require receivers to send explicit feedback to the sender, since joins and leaves can often be processed at downstream routers. Another attractive feature of this general approach is that receivers can act autonomously, i.e. receivers need not coordinate join attempts with one another. These features of the congestion control algorithm are particularly important when integrating with the digital fountain approach to reliability in which receiver-to-source and inter-receiver communication are undesirable. We refer the reader to [32] for further details, including evaluation of the TCP-friendliness of this scheme.

B. Scheduling Transmissions Across Multiple Multicast Groups

As described earlier, a receiver at level i subscribes to *all* layers 0 through i. Therefore, when using codes with a constant stretch factor c, it is important to schedule packet transmissions carefully across the multiple layers, so as to minimize the number of duplicate packets that a receiver receives. In our previous work [8], we provided heuristics for scheduling packets from a finite Tornado encoding across multiple layers, following the work of Bhattacharyya et al. [3], who demonstrated that a packet scheduling scheme for cumulative layered multicast exists whereby the sender transmits a permutation of the entire set of encoding packets on any given set of cumulative layers before repeating a packet.

However, the LT codes make this scheduling consideration obsolete, as the encoding process is on-the-fly and memoryless, thus these codes can naturally be employed with a memoryless scheduling process, where any packet on any given layer is constructed independently from any other packet. In brief, this eliminates correlations between packets across layers, and any subset of packets of a given size is equally likely to restitute the source file, regardless of which layers those packets were transmitted on.

C. Reconstruction at the Receiver

As detailed in the previous subsection, the receiver is responsible for performing receiver-driven joins and leaves to facilitate congestion control. The other activity that the receiver must perform is the reconstruction of the source data. There are two ways to implement the receiver decoding protocol. The first is an incremental approach in which the receiver performs preliminary decoding operations after each packet arrives. This approach leads to some redundant computation: reconstructed source data may later arrive intact. Moreover, there may be a modest overhead in processing individual packets immediately on arrival. A second, patient approach that reduces these effects is to wait until a fixed number of packets arrive from which it is likely that the source can be reconstructed, based on statistical observations. If the decoding cannot be completed at this time, then additional packets may be processed individually or in small groups. While the incremental approach has the benefit of enabling some decoding computation to be overlapped with packet reception, we found the patient approach to be simpler to implement in practice, with little loss of decoding speed. In the Tornado Z implementation we describe, we wait until $1.06k$ packets arrive[2], attempt to decode, and then process additional packets individually as needed until decoding is successful.

D. Experimental Setup and Results

Now we turn to measurements of the efficiency of our experimental system. First, we clarify the two sources of inefficiency. Recall that the *decoding inefficiency*, $1 + \epsilon = \eta_c$, captures the inefficiency due specifically to our use of sparse codes. It is defined as

$$\eta_c = \frac{\#\text{ of distinct packets rcvd before reconstruction}}{\#\text{ of source data packets}}.$$

There is, however, another possible source of inefficiency: a receiver could obtain duplicate packets. The *distinctness inefficiency*, η_d, captures the loss in efficiency caused by receiving duplicate packets. This can occur either by cycling

[2] This quantity is carefully chosen based on statistical observations and depends on both the code used and the file size.

through the carousel under high loss rates by temporarily suspending the transfer, or by changing the receiver subscription layer as described in Section VII-B. It is defined as

$$\eta_d = \frac{\text{Total \# of packets received}}{\text{\# of distinct packets received}}.$$

Combining these two effects yields the *reception inefficiency*, η. It is defined as

$$\eta = \frac{\text{\# of packets received prior to reconstruction}}{\text{\# of source data packets}}.$$

It is clear that $\eta = \eta_c \eta_d$.

The experimental results measure our prototype implementation. Besides testing the layered protocol we have described, we also test a single layer protocol. That is, we also measure the reception inefficiency when the server transmits the file on a single multicast group at a fixed rate. These results allow us to focus on the efficiency of the packet transmission scheme independent of the layering scheme for congestion control. In both cases the server encodes using Tornado Z to produce the encoding. The server runs two threads: a UDP unicast thread that provides various control information such as multicast group information and file length to the receiver and a multicast transmission thread. For both protocols, the receivers connect to the server's known UDP port for control information and on receipt of the information, subscribe to the appropriate multicast groups.

Our test source data consisted of a Quicktime movie (a clip available from `www.nfl.com`) with size slightly over two megabytes. The encoding algorithm used a stretch factor of $c = 2$ to produce 8264 packets of size 500 bytes. The packets were additionally tagged with 12 bytes of information (packet index, serial number and group number) to give a final packet size of 512 bytes. The server and receivers were on three different subnets, located at Berkeley, CMU and Cornell. There were 16 hops on the path from Berkeley to CMU, and the bottleneck bandwidth (obtained by using *mtrace* and *pathchar* [14]) was 8 Mb/s with an RTT of 60 ms. There were 17 hops on the path from Berkeley to Cornell, and the bottleneck bandwidth was 9.3 Mb/s with an RTT of 87 ms. The base layer bandwidth was set to a rate ranging from 64 Kb/s to 512 Kb/s. We ran experiments with the server both at Berkeley and at CMU and with the receivers located at the other two subnets. Locating the server at CMU tended to generate higher packet loss rates for the same transmission bandwidth. The machines used at all three sites were running Solaris 2.5.1. When running the layered protocol, we used 4 layers.

In our initial experiments, in some cases we witnessed loss rates over the course of the transmission of nearly 20% – rates that are admittedly far higher than the congestion control techniques of [32] were intended to handle. To generate even higher loss rates that might arise in other environments, such as mobile wireless networks, we turned off congestion control and set the base layer rate artificially high, causing a router within our LAN to drop packets persistently.

The data from the two sets of experiments are shown in Figure 6. As seen from the graphs for the single layered case, for packet losses of less than 50%, the distinctness inefficiency is almost always 1, as is to be expected. Thus, for low loss rates, the reception inefficiency is effectively the decoding inefficiency, which in our example was roughly 1.07 on average. (This decoding inefficiency is slightly higher than for Tornado Z because a slightly different code was used in these experiments, and because we wait until at least $1.06k$ packets arrive before trying to decode.) We further observe that the transmission scheme is robust even under severe loss rates - even at loss rates approaching 50%, the reception inefficiency is generally below 1.4. This reception inefficiency can be mitigated either by the use of Tornado codes with a larger stretch factor or by the use of LT codes.

Figure 6 also shows experimental data for the multilayered case. We observe that the use of multiple layers for congestion control increases the distinctness inefficiency. This is natural as switching among subscription levels can cause the receiver to receive packets that had already been obtained at other subscription levels. For high loss rates, the distinctness inefficiency remained low because receivers generally subscribed only to the base layer. Again this cost can be mitigated with the use of LT codes.

VIII. Conclusion

The introduction of fast erasure codes yields significant new possibilities for the design of reliable multicast protocols. To explore these possibilities, we formalized the notion of an ideal digital fountain and explained how Tornado codes and LT codes can yield a much closer approximation to a digital fountain than previous systems based on standard Reed-Solomon erasure codes. The prototype multicast data distribution system which we built demonstrates that simple protocols using Tornado codes are effective in practice.

Given that we can closely approximate a digital fountain with Tornado and LT codes, we conclude with other possible applications for such encoding schemes. One application is dispersity routing of data from endpoint to endpoint in a packet-routing network. With packets generated by a digital fountain, the source can inject packets along multiple paths in the network. Those packets that experience congestion are delayed, but the destination can recover the data once a sufficient number of packets arrive, irrespective of the paths they took. This application dates back to the seminal works on dispersity routing by Maxemchuk [20] and information dispersal by Rabin [27]. Both suggested using standard erasure codes, but we expect that faster codes and the digital fountain approach will lead to improved practical dispersity routing schemes.

Related applications which we have considered include downloading content in parallel from multiple mirror sites [7] and content delivery in overlay networks such as peer-to-peer networks [5]. By encoding the content, clients, servers, and peers are freed from complex negotiations that arise when all of the individual, unencoded packets from

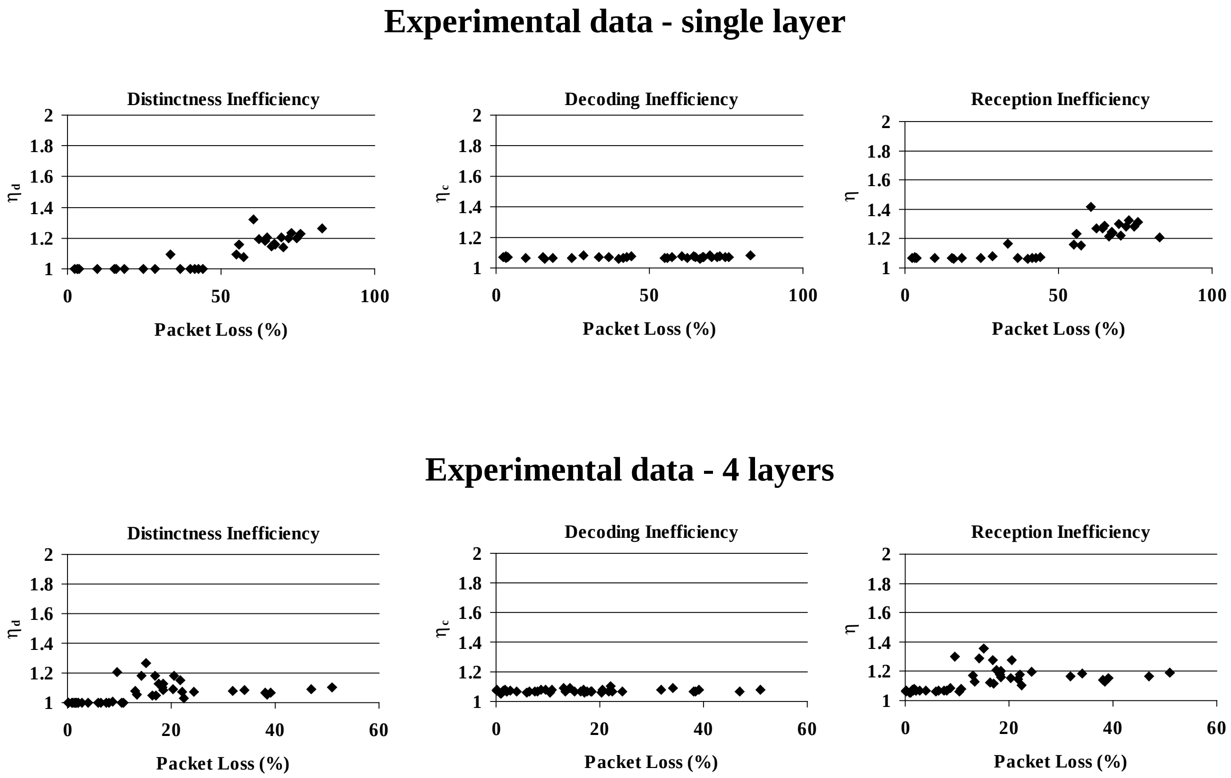

Fig. 6. Experimental Results of the Prototype

the source file must be collected across heterogeneous end-to-end connections. Instead, by using the digital fountain paradigm, receivers can draw encoded content from servers or peers in parallel until they receive sufficiently many encoded packets to reconstruct the file.

Acknowledgments

We thank Lisa Camesano, Sally Floyd and the anonymous SIGCOMM '98 reviewers for their helpful comments on an earlier version of this work, as well as Don Towsley and the anonymous J-SAC referees for their feedback.

References

[1] S. Acharya, M. Franklin, and S. Zdonik, "Dissemination Based Data Delivery Using Broadcast Disks," *IEEE Personal Communications*, December 1995, pp. 50-60.

[2] A. Bestavros. "AIDA-Based Real-Time Fault-Tolerant Broadcast Disks." In *Proceedings of the 16th IEEE Real-Time System Symposium,* June 1996.

[3] S. Bhattacharyya, J. F. Kurose, D. Towsley, and R. Nagarajan, "Efficient Rate-Controlled Bulk Data Transfer using Multiple Multicast Groups." In *Proc. of INFOCOM '98*, San Francisco, April 1998, pp. 1172-1179.

[4] J. Blömer, M. Kalfane, M. Karpinski, R. Karp, M. Luby, and D. Zuckerman, "An XOR-Based Erasure-Resilient Coding Scheme," *ICSI Technical Report No. TR-95-048*, August 1995.

[5] J. Byers, J. Considine, M. Mitzenmacher, and S. Rost, "Informed Content Delivery Across Adaptive Overlay Networks." In *Proc. of ACM SIGCOMM '02*, Pittsburgh, PA, August 2002.

[6] J. Byers, G. Horn, M. Luby, M. Mitzenmacher, and W. Shaver, "FLID-DL: Congestion Control for Layered Multicast," in *IEEE Journal on Selected Areas in Communications, special issue on Network Support for Multicast Communications*, November 2002. Preliminary version appears in *Proc. of NGC 2000*, Stanford, CA, November 2000, pp. 71-81.

[7] J. Byers, M. Luby, and M. Mitzenmacher, "Accessing Multiple Mirror Sites in Parallel: Using Tornado Codes to Speed Up Downloads." In *Proc. of IEEE INFOCOM '99*, New York, NY, March 1999, pp. 275-283.

[8] J. Byers, M. Luby, M. Mitzenmacher and A. Rege, "A Digital Fountain Approach to Reliable Distribution of Bulk Data." In *Proc. of ACM SIGCOMM '98*, Vancouver, August 1998, pp. 56-67.

[9] Digital Fountain Technology Overview, `URL = http://www.digitalfountain.com/technology`, Digital Fountain, Inc., August 2001.

[10] S. Floyd, V. Jacobson, C. G. Liu, S. McCanne, and L. Zhang, "A Reliable Multicast Framework for Light-Weight Sessions and Application Level Framing." In *ACM SIGCOMM '95*, pp. 342-356, August 1995.

[11] J. Gemmell, "ECSRM – Erasure Correcting Scalable Reliable Multicast," *Microsoft Research Technical Report MS-TR-97-20*, June 1997.

[12] C. Huitema, "The Case for Packet Level FEC." In *Proc. of IFIP 5th Int'l Workshop on Protocols for High Speed Networks,* Sophia Antipolis, France, October 1996.

[13] Cauchy-based Reed-Solomon codes. Available at `http://www.icsi.berkeley.edu/~luby`.

[14] V. Jacobson, `pathchar`, available at `http://www-nrg.ee.lbl.gov/pathchar`.

[15] S. K Kasera, J. Kurose, and D. Towsley, "A Comparison of Server-based and Receiver-based Local Recovery Approaches for Scalable Reliable Multicast," in *Proc. of IEEE INFOCOM '98*, pp. 988-995, March 1998.

[16] M. Klamkin and D. Newman, "Extensions of the Birthday Surprise," Journal of Combinatorial Theory, Volume 3, pp. 279-282, 1967.

[17] Michael Luby, "Information Additive Code Generator and Decoder for Communication Systems", U.S. Patent No. 6,307,487, October 23, 2001.

[18] M. Luby, M. Mitzenmacher, A. Shokrollahi, and D. Spielman, "Efficient Erasure Correcting Codes." *IEEE Transactions on Information Theory*, 47(2), pp. 569-584, February 2001. A preliminary version entitled "Practical Loss-Resilient Codes" appeared in *ACM Symposium on Theory of Computing*, April 1997.

[19] M. Luby, M. Mitzenmacher, and A. Shokrollahi, "Analysis of Random Processes via And-Or Tree Evaluation." In *Proceedings of the* 9^{th} *Annual ACM-SIAM Symposium on Discrete Algorithms*, January 1998, pp. 364–73.

[20] N. F. Maxemchuk, "Dispersity Routing in Store and Forward Networks." Ph. D. thesis, University of Pennsylvania, May 1975.

[21] S. McCanne, V. Jacobson, and M. Vetterli, "Receiver-driven Layered Multicast." In *Proc. of ACM SIGCOMM '96*, pp. 117-130, August 1996.

[22] J. Nonnenmacher and E. W. Biersack, "Reliable Multicast: Where to Use Forward Error Correction." In *Proc. of IFIP 5th Int'l Workshop on Protocols for High Speed Networks,* Sophia Antipolis, France, October 1996, pp. 134-148.

[23] J. Nonnenmacher and E.W. Biersack, "Asynchronous Multicast Push: AMP." In *Proc. of International Conference on Computer Communications*, Cannes, France, November 1997, pp. 419-430.

[24] J. Nonnenmacher, M. Lacher, M. Jung, G. Carl, and E.W. Biersack, "How Bad is Reliable Multicast Without Local Recovery?" In *Proc. of INFOCOM '98*, San Francisco, April 1998, pp. 972-979.

[25] J. Nonnenmacher, E. W. Biersack, and D. Towsley, "Parity-Based Loss Recovery for Reliable Multicast." IEEE/ACM Transactions on Networking, 6(4), pp. 349-361, Aug. 1998. Preliminary version appeared in *Proc. of ACM SIGCOMM '97.*

[26] S. Paul, K. Sabnani, J. Lin, and S. Bhattacharyya, "Reliable Multicast Transport Protocol (RMTP)." IEEE Journal on Selected Areas in Communication, Special Issue on Network Support for Multipoint Communication, 15(3), pp. 407–42, April 1997.

[27] M. O. Rabin, "Efficient Dispersal of Information for Security, Load Balancing, and Fault Tolerance." *Journal of the ACM*, Volume 36(2), pp. 335-348, April 1989.

[28] L. Rizzo, "Effective Erasure Codes for Reliable Computer Communication Protocols." In *Computer Communication Review*, 27(2), April 1997, pp. 24-36.

[29] L. Rizzo and L. Vicisano, "A Reliable Multicast data Distribution Protocol Based on Software FEC Techniques." In *Proc. of HPCS '97*, Greece, June 1997.

[30] D. Rubenstein, S. Kasera, D. Towsley, and J. Kurose, "Improving Reliable Multicast Using Active Parity Encoding Services," in *Proc. of IEEE INFOCOM '99*, March 1999, pp. 1248-1255.

[31] E. Schooler and J. Gemmell, "Using Multicast FEC to Solve the Midnight Madness Problem," *Microsoft Research Technical Report MS-TR-97-25*, September 1997.

[32] L. Vicisano, L. Rizzo, and J. Crowcroft. "TCP-like congestion control for layered multicast data transfer." In *Proc. of INFOCOM '98*, San Francisco, March 1998, pp. 996-1003.

[33] M. Yajnik, J. Kurose, and D. Towsley, "Packet Loss Correlation in the MBone Multicast Network." In *Proceedings of IEEE Global Internet '96,* London, November 1996, pp. 94-99.

[34] R. Yavatkar, J. Griffoen and M. Sudan, "A Reliable Dissemination Protocol for Interactive Collaborative Applications." In *Proceedings of ACM Multimedia '95,* San Francisco, 1995, pp. 333-344.