A Digital Fountain Approach to Asynchronous Reliable Multicast

John W. Byers * Michael Luby† Michael Mitzenmacher‡

September 6, 2001

Abstract

The proliferation of applications that must reliably distribute bulk data to a large number of autonomous clients 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 clients to acquire bulk data 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.

1 Introduction

A natural solution for companies that plan to efficiently disseminate large, rich content over the Internet to millions of concurrent users is multicast or broadcast transmission. These transmissions must be fully reliable, have low network

*Boston University and Digital Fountain, Inc. Research initiated while the author was at UC Berkeley and the International Computer Science Institute, Berkeley, California and was supported in part by National Science Foundation operating grant NCR-9416101. **Email:** byers@cs.bu.edu

†Digital Fountain, Inc. Research initiated while the author was at the International Computer Science Institute, Berkeley, California and was supported in part by National Science Foundation operating grant NCR-9416101. **Email:** luby@digitalfountain.com

‡Harvard University and Digital Fountain, Inc. Research initiated while the author was at Digital Systems Research Center, Palo Alto, California. Supported in part by National Science Foundation operating grant CCR-9983832. **Email:** michaelm@cs.harvard.edu. A preliminary version of this paper appeared in [7].

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. These applications require more than just a reliable multicast protocol, since users 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 users.

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 clients. As clients 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 clients. More sophisticated solutions that address these limitations by using techniques such as local repair, polling, or the use of a hierarchy have been proposed [9, 16, 22, 23, 37], but these solutions as yet appear inadequate [26]. 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[1] to reliable multicast [10, 24, 25, 27, 30, 31, 33, 35]. 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 [27] 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 [11, 23, 24, 25, 30, 31, 33]. The limitation of these codes is that encoding and decoding times are slow on large block sizes, effectively limiting k to small values for practical applications. Hence, their

[1]Erasure codes are sometimes called forward-error correcting codes (FEC codes) in the networking community. However, FEC often refers to codes that detect and correct errors, and these codes are typically implemented in special purpose hardware. To avoid confusion, we always refer to the codes we consider as erasure codes.

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 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. Their 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 [31, 33], 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 6.

The approaches described above that eliminate 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 is similar to ideas found in the seminal works of Maxemchuk [20, 21] and Rabin [29]. 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 to the Internet, satellite networks, and wireless networks with mobile agents. 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 3, we describe the digital fountain solution. In Section 4, 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 5, 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. [8]. 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 6. In Section 7, 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 [35] and others. We conclude with additional research directions we are pursuing which instantiate the digital fountain approach for other content distribution methods.

Note to myself: Add SSM somewhere in here.

2 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:** The file is guaranteed to be delivered in its entirety to all receivers.

- **Efficient:** Both the total number of packets each receiver 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 receiver should be no more than it would have been had point-to-point connections been used.

- **On demand:** 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.

- **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, efficient, and on demand implies that receiver robustness to missing packets is crucial. For example, a receiver may sporadically be interrupted, continuing the download several times before completion. During the interruptions the server will continue sending out a stream of packets 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.

3 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 measure a new approach that implements an approximation to this ideal solution that is superior to previous approaches.

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.* 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.

4 Building a Digital Fountain with 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.

We note that 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 $s = n/k$ 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. We emphasize that while a preset stretch factor is commonly used in practice, it is not truly necessary when using some erasure codes, including Reed-Solomon codes, for example. A suitable implementation of Reed-Solomon codes could generate unique encoding packets *on-the-fly*, creating new encoding packets on demand as needed. Technically, the encoding packets generated would be distinct from previously generated encoding packets only with high probability, as each would necessarily be chosen from a large, but finite universe of encoding packets.

Although a finite stretch factor limits the extent to which such erasure codes can approximate a digital fountain, a reasonable approximation proposed by other researchers (e.g., [25, 30, 31, 35]), is to set n to be a multiple of k, then repeatedly cycle through transmission of these n encoding packets. There are several 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 transmission 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 **??** is when a cumulative layered multicast scheme is used to serve heterogeneous receivers with different transfer rates. Slow receivers receive only 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 parallel downloading, where a source may be downloading encoding packets from several receivers. A finite stretch factor requires proper scheduling to reduce the incidence of duplicate transmissions, as explained in [6]. For all of these reasons, systems where encoding packets can be generated on-the-fly may be preferable in many situations.

A further key consideration is 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 first alternative we propose is to avoid this cost by using the much faster Tornado codes [18]. As always, there is a tradeoff associated with using one code in place of another. 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 6. We also study a second alternative, LT codes. Whereas Tornado codes admit only moderate stretch factors and have a decoding time that depends on n, LT codes have the potential to generate effectively unbounded stretch factors *and* can be decoded in time depending only on k (and not n). But before considering these performance implications in full detail, we first provide a description of the constructions used to build these codes and their properties.

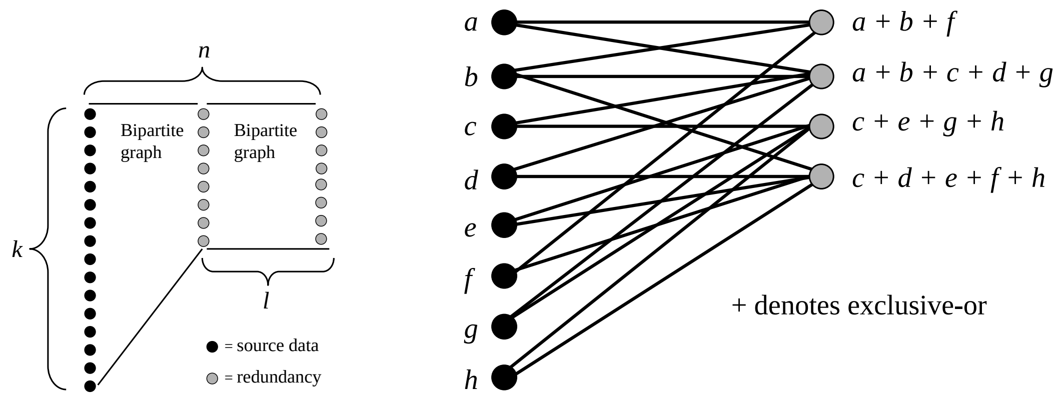

Figure 1: Structure of Tornado Codes

5 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 many respects to Tornado codes, but exhibit additional key properties which provide a better approximation to an idealized digital fountain. 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 [18, 19] 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 ℓ redundant packets for a total of $n = k + \ell$ 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 x_i, and we think of the jth redundant packet as containing the value of a variable y_j that is a linear combination of the x_i 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 $y_3 = x_1 + x_2\alpha + \ldots + x_k\alpha^{k-1}$, 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.

We emphasize again that Reed-Solomon codes need not rely on a pre-determined stretch factor. If a new packet is desired, the sender can choose an α from the field and generate an appropriate packet. This can be done in time $O(k)$. This requires some agreed method of determining the field value α (such as pseudo-randomly using bits from the packet sequence number, for example). The number of different field elements depends on the number of bits in the packet, but one can easily establish a setting where the potential number of encoding packets is extremely large, at the possible expense of computation time for encoding and decoding.

Reed-Solomon 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 Reed-Solomon 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. *Mike L.: Check the next sentence and/or remind me of the reference.* Additional theoretical work has demonstrated that faster asymptotic decoding times are possible, but they continue to perform much more slowly than the quadratic algorithms even when k and ℓ are quite large.

The large decoding time for Reed-Solomon codes arises from the *dense* system of linear equations used. Both Tornado and LT 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. 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 [18, 19].

For 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. For both of these codes, the number of equations of various forms are carefully chosen in advance; in particular, the number of encoding packets n must be given as an input parameter, so Tornado codes do require an underlying strecth factor. The encoding time is dominated by the number of exclusive-or operations in the system of equations. The matrices underlying linear transformations of this type are referred to as *parity-check matrices*.

The decoding process for both Tornado and LT 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 $y_3 = x_1 \oplus x_4 \oplus x_7$. Suppose the redundant packet containing y_3 has been received, as well as the source data packets containing x_1 and x_7, but x_4 has not been received. Then we can solve for x_4 using only exclusive-or operations by setting $x_4 = y_3 \oplus x_1 \oplus x_7$. 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 derivation 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 FEC 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.

For the fast FEC codes, we say that the *decoding inefficiency* is $1+\epsilon$ if $(1+\epsilon)k$ encoding packets are required to reconstruct the source data on average. For Tornado and LT 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 the fast FEC codes over standard codes is that they trade off a small increase in decoding inefficiency for a substantial decrease in encoding and decoding times. Recall that Reed-Solomon codes have encoding times proportional to $k\ell P$ and decoding times proportional to keP. As a result, Reed-Solomon codes can only be applied in practice when k and ℓ are relatively small. (Values used in [27, 31, 35, 33] have k and ℓ ranging from 8 to 256.) In contrast, the Tornado codes we consider have encoding and decoding times that are proportional to $(k + \ell)\ln(1/\epsilon)P$ with decoding inefficiency $1 + \epsilon$. And in practice, the encoding and decoding times of these codes are orders of magnitude faster than Reed-Solomon codes for large values of k and ℓ. A summary comparing the properties of file reconstruction when Tornado codes when used with a fixed stretch factor compared against standard Reed-Solomon codes is given in Table 1.

Tornado codes possess the drawbacks that a stretch factor must be determined ahead of time and that decoding takes time proportional to the size of the stretched encoding. As we have suggested previously, for situations where there are substantial loss rates or when the receiver may receive only a fraction of packet transmissions, this limitation may make Tornado codes unsuitable. The LT codes, similar in style to Tornado codes, avoid this significant limitation [**?**]. Like Reed-Solomon codes, LT codes have a structure that allows new encoding packets to be produced on-the-fly from an extremely large set of possibilities. Unlike Reed-Solomon codes, with LT codes new encoding packets

	Tornado	Reed-Solomon
Decoding inefficiency	$1+\epsilon$ required	1
Encoding times	$(k+\ell)\ln(1/\epsilon)P$	$k\ell P$
Decoding times	$(k+\ell)\ln(1/\epsilon)P$	keP
Basic operation	XOR	Field operations

Table 1: Properties of Tornado vs. Reed-Solomon codes when a fixed stretch factor is employed

	LT	Reed-Solomon
Decoding inefficiency	$1+\epsilon$ required	1
Per packet encoding times	$\ln(1/\epsilon)P$	kP
Decoding times	$k\ln(1/\epsilon)P$	keP
Basic operation	XOR	Field operations

Table 2: Properties of LT vs. Reed-Solomon on-the-fly codes.

can be constructed in time $O(\ln(1/\epsilon))$ on average. Properly constructed, LT codes also hold other advantages over Tornado codes, such as lower variance in the number of encoding packets that need to be obtained before decoding can complete. With LT codes, therefore, one obtains a closer approximation to an idealized digital fountain. A summary comparing the properties of the per-packet encoding costs and total decoding costs of on-the-fly LT codes when compared with Reed-Solomon codes is given 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 Reed-Solomon codes.

5.2 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 levelled graph, as depicted in Figure 1. The nodes of the leftmost level of the graph correspond to the source data. Subsequent levels contain the redundant data. Each redundant packet is the exclusive-or of the packets held in the neighboring nodes in the level 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 levels. 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 levels; the number of nodes in the levels are 16,000, 8,000 and 8,000 respectively.

The graph between the first two levels 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 level 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 level. 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.[2] The distribution of node degrees on the right 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 level, 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 second layer uses a graph with 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 edges 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.

5.3 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 [10, 27, 30, 31, 33, 35]. The erasure

[2] 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 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 3: 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 4: Comparison of decoding times.

code listed in Tables 3 and 4 as *Cauchy* [4] is a standard implementations of Reed-Solomon erasure codes based on Cauchy matrices. (We note that the Cauchy implementation, available at [12], is faster for larger values of k than the implementation of Reed-Solomon codes based on Vandermonde matrices by Rizzo [30].) 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 reflect the appropriate high level behavioral difference between Reed-Solomon 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.

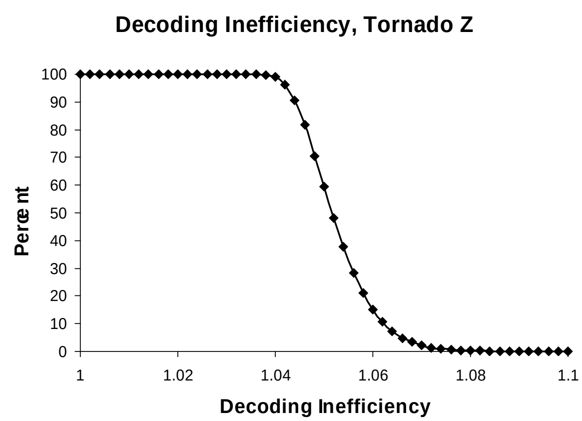

Figure 2: Decoding inefficiency variation over 10,000 trials of Tornado Z.

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 encoded and decoded much faster than Reed-Solomon 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 clients could reconstruct the source data before receiving this many packets.

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 5.2. 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.

6 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 [27, 30, 31, 33]. 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 $k+\ell$ packets using a standard 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 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 true for both Tornado and LT 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 $1+\epsilon$ if one must obtain $(1+\epsilon)k$ distinct packets in order to decode the source data. For both Tornado and LT codes, there is some decoding inefficiency based on 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 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.

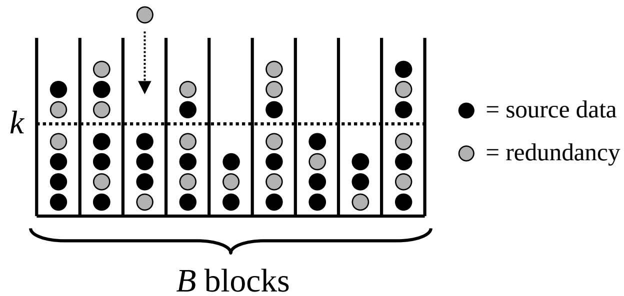

Figure 3: Waiting for the last blocks to fill...

- 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 times and decoding inefficiency.

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 packets in a random order from the source end. 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 [7], we provided results from trace-driven simulations of Internet traffic taken from [36]. While there is some small effect, our results from trace-driven simulations suyggest that overall the effect of burstiness is small enough to discount for comparison, so we focus on results with the random loss model in what follows.

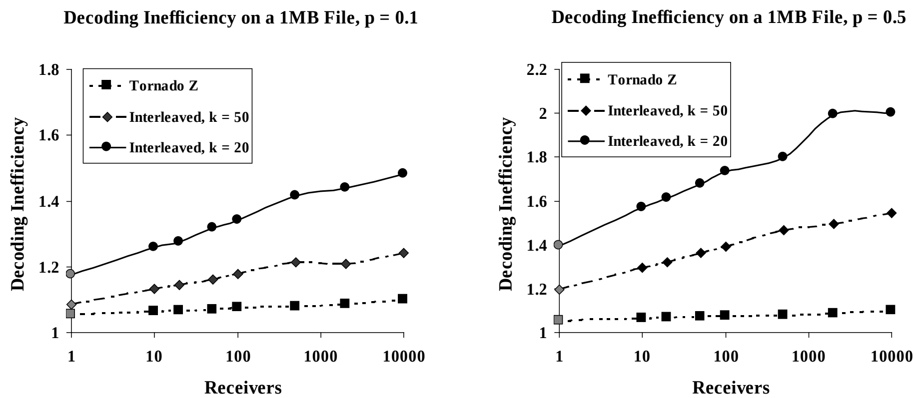

Figure 4: Comparison of decoding inefficiency for codes with comparable decoding times.

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 1% of the time. In Table 5, 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 5 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 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 4, 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 4.

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 5: Speedup of Tornado Z codes over interleaved codes with comparable efficiency.

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), 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.

6.2 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 [27],

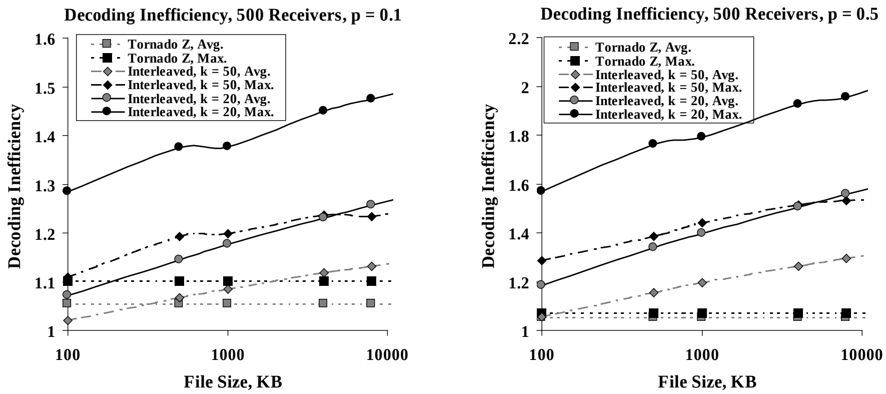

Figure 5: Comparison of decoding inefficiency as file size grows.

interleaved codes and Tornado 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 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.

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 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.

7 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 [22, 25] 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 [35]. In a companion paper [5], 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 FEC 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 client uses to decode the message. Then we describe the experimental setup and performance results of our system.

7.1 Integrating with Layered Congestion Control Methods

The approach to accommodating receiver heterogeneity with appropriate congestion control mechanisms which we propose follows the lead of other authors who advocate *layered* multicast [22, 25, 35]. 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* in that a receiver subscribing to layer i also subscribes to all layers beneath it. 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 [35] 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 clients, 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 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 [35] for further details, including their evaluation of the TCP-friendliness of this scheme.

7.2 Scheduling Packet 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 client receives. In our previous work [**?**], 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.

We find that this choice is suitable in practice because

However, the LT codes make this scheduling consideration obsolete, as the encoding process is on-the-fly and memoryless, thus they 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.

7.3 Reconstruction at the Client

As detailed in the previous subsection, the client is responsible for performing receiver-driven joins and leaves to facilitate congestion control. The other activity that the client must perform is the reconstruction of the source data. There are two ways to implement the client decoding protocol. The first is an incremental approach in which the client 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 $1.060k$ packets arrive[3] attempt to decode, and then process additional packets individually as needed until decoding is successful.

7.4 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 the use of Tornado codes. It is defined as

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

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 through the carousel under exceedingly high loss rates or by changing the receiver subscription layer as described in Section 7.2. 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

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

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

For high loss rates, the distinctness inefficiency remained low because receivers generally subscribed only to the base layer. An interesting direction we intend to pursue further is to study how the reception efficiency varies with the rates of change in receiver subscription level.

8 Conclusion

The introduction of fast forward error correcting 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. Additional

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

wide-area trials conducted by Digital Fountain, Inc. over the course of several years and for a wide variety of clients confirm the effectiveness of the approach.

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, 21] and information dispersal by Rabin [29]. Both suggested using standard erasure codes, but we expect faster codes and the digital fountain approach will lead to improved practical dispersity routing schemes.

A related application which we have considered is that of downloading content in parallel from multiple mirror sites [6]. By encoding the content, clients and servers are freed from complex negotiations that arise when all of the individual, unencoded packets from the source file must be collected across heterogeneous end-to-end connections. Instead, by using the digital fountain paradigm, clients can draw encoded content from sites in parallel without any additional coordination, and close all streams once they can restitute the file. Perhaps the most challenging problem for this application is orthogonal to the use of codes: how best to *select* the servers from which to conduct the download?

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 *Procedings of the 16th IEEE Real-Time System Symposium,* 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.

[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, M. Frumin, G. Horn, M. Luby, M. Mitzenmacher, A. Roetter, and W. Shaver, "FLID-DL: Congestion Control for Layered Multicast." In *Proc. of NGC 2000*, Stanford, CA, November 2000, pp. 71-81.

[6] 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.

[7] 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.

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

[9] 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.

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

[11] 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.

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

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

[14] S. K Kasera, J. Kurose, and D. Towsley, "A comparison of server-based and receiver-based local recovery approaches for scalable reliable multicast," in *IEEE INFOCOM 1998*, pages 988-995.

[15] M. S. Lacher, J. Nonnenmacher, and E. W. Biersack, "Performance comparison of centralized versus distributed error recovery for reliable multicast," *IEEE/ACM Transaction on Networking*, 8(2):224-238, 2000.

[16] 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.

[17] M. G. Luby, "Some really long technical patent title goes here," The patent has an official number. It will issue soon.

[18] M. Luby, M. Mitzenmacher, A. Shokrollahi, D. Spielman, and V. Stemann, "Practical Loss-Resilient Codes." In *Proceedings of the* 29^{th} *ACM Symposium on Theory of Computing*, 1997. **Cite the journal version.**

[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.

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

[21] N. F. Maxemchuk, "Dispersity Routing." Proceedings of ICC '75, San Francisco, CA, pp. 41-10 – 41-13, 1975.

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

[23] C. K. Miller, "Reliable Multicast Protocols: A Practical View." In *Proc. of the 22nd Annual Conference on Local Computer Networks (LCN '97)*, 1997.

[24] 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,* pp. 134-148, Sophia Antipolis, France, October 1996. Chapman and Hall.

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

[26] 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.

[27] 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.*

[28] C. Papadopoulos, G. Parulkar, and G. Varghese, "An error control scheme for large-scale multicast applications," in *IEEE INFOCOM 1998*, pages 1188-1196.

[29] M. O. Rabin, "Efficient Dispersal of Information for Security, Load Balancing, and Fault Tolerance." *Journal of the ACM*, Volume 38, pp. 335-348, 1989.

[30] L. Rizzo, "Effective Erasure Codes for Reliable Computer Communication Protocols." In *Computer Communication Review*, April 1997.

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

[32] D. Rubenstein and D. Towsley, "Improving reliable multicast using active parity encoding services," in *IEEE INFOCOM 1999*.

[33] E. Schooler and J. Gemmell, "Using multicast FEC to solve the midnight madness problem," *Microsoft Research Technical Report MS-TR-97-25*, September 1997.

[34] D. Towsley, J. Kurose, and S. Pingali. "A comparison of sender-initiated and receiver-initiated reliable multicast protocosl," in *IEEE Journal of Selected Areas in Communications*, vol. 15, pp. 398-406, 1997.

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

[36] 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.

[37] 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.