A Digital Fountain Approach to Reliable Distribution of Bulk Data

John W. Byers [*] **Michael Luby** [†]

Michael Mitzenmacher [‡] **Ashutosh Rege** [§]

TR-98-005

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 a new class of erasure codes that 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.

[*] UC Berkeley and International Computer Science Institute, Berkeley, California. Research supported in part by National Science Foundation operating grant NCR-9416101. Contact author: byers@cs.berkeley.edu

[†] International Computer Science Institute, Berkeley, California. Research supported in part by National Science Foundation operating grant NCR-9416101.

[‡] Digital Systems Research Center, Palo Alto, California.

[§] International Computer Science Institute, Berkeley, California. Research supported in part by National Science Foundation operating grant NCR-9416101.

Keywords: digital fountain, reliable data distribution, bulk distribution, on demand download, erasure codes, forward-error correcting (FEC), IP multicast, broadcast, lossy channels, heterogeneous conditions.

1 Introduction

Software companies that plan to efficiently disseminate new software over the Internet to millions of users simultaneously will require multicast or broadcast transmission [18]. These transmissions must be fully reliable, have low network overhead, and support vast numbers of receivers with heterogeneous characteristics. Other activities which have similar requirements include distribution of video and financial information, database replication, and popular web site access. These applications require more than just a reliable multicast protocol, since users wish to access the data at times of their choosing and these 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 analog of this solution is unscalable. For example, consider a video server distributing a movie over the Internet 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 which address these limitations by using techniques such as local repair, polling, or the use of a hierarchy have been proposed [3, 7, 10, 11, 21], but these solutions as yet appear inadequate. Moreover, whereas 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 retransmission have led many researchers to consider applying Forward Error Correction based on erasure codes[1] to reliable multicast [4, 12, 13, 14, 16, 17, 18, 19]. 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 recent work of Nonnenmacher, Biersack and Towsley [14] 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 [5, 11, 12, 13, 16, 17, 18]. The limitation of these codes is that encoding and decoding times are slow, effectively limiting k to small values for practical applications. Hence, their solution involves breaking the source data into smaller blocks of packets and encoding over these blocks. Receivers which 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

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

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 [17, 18], adding redundant codewords to the carousel can dramatically reduce reception overhead. Using Reed-Solomon codes, these papers advocate adding a fixed amount of redundancy to blocks of the transmission. 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 transmissions, especially while waiting for the last packets from the last few blocks it needs to reconstruct.

The approaches described above that eliminate the need for retransmission requests can be thought of as weak 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. Our approach is to construct better approximations of a digital fountain as a basis for protocols that perform reliable distribution of bulk data.

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, a new class of erasure codes that have extremely fast encoding and decoding algorithms. These codes 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. Finally, 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 [19]. We conclude with additional research directions for the digital fountain approach.

2 Requirements for an Ideal Protocol

We recall an example application in which millions of clients want to download a new release of software over the course of several days. This software download application highlights several important features common to many similar applications which must distribute bulk data. In addition to keeping network traffic to a minimum, a scalable protocol for distributing the software using multicast should be:

- **Reliable:** The file is guaranteed to be delivered in its entirety to all receivers.
- **Efficient:** The time to download the file and any additional processing time should incur minimal overhead. Ideally, the end-user should not be able to distinguish between service over a multicast group and service over a point-to-point connection.
- **On demand:** Clients may initiate the download at their discretion, implying that different clients may start the download at widely disparate times. Clients may sporadically interrupt 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 and data rates.

We also state our assumptions regarding channel characteristics. IP multicast on the Internet, satellite transmission, wireless transmission, and cable transmission are representative of channels we consider. Perhaps the most important property of these channels is that the return feedback channel from the clients to the server is typically of limited capacity, or is non-existent. 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 are used to correct and detect errors within a packet. But if a packet has too many errors to correct, it is discarded and treated as a loss.

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 an experimental system which 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, which constitute an encoding of the source data. The server will transmit the encoding packets whenever there are any clients listening in on the multicast 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 for quenching thirst: drinking a glass of water, irrespective of the particular drops that fill the glass, quenches one's thirst. The digital fountain protocol 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. However, erasure codes are designed to stretch a file consisting of k packets into n encoding packets, where both k and n are input parameters. For the remainder of the paper, we refer to the ratio n/k as the *stretch factor* of an erasure code. This limits the extent to which an erasure code can achieve the properties of a digital fountain.

An obvious way to approximate a digital fountain also proposed by other researchers (e.g., [13, 16, 17, 19]) is to set n to be a multiple of k, and repeatedly cycle through and send the n encoding packets. Then, the client is guaranteed to be able to recover the original file as long as k distinct packets are received. However, as detailed in subsequent sections, the encoding and decoding processing times for standard erasure codes are prohibitive even for moderate values of k and n. We avoid this cost by using the much faster Tornado codes.

Throughout the paper we set n to be a small multiple of k, in particular $n = 2k$, in order to keep the processing and memory requirements for the erasure codes reasonable. The disadvantage of choosing such a small value for n is that under sufficiently high loss rates a client may not receive k out of n packets in one cycle. Therefore a client may receive useless duplicate packets, which decreases the channel efficiency. Our experimental results indicate that while this degradation is non-negligible under very high loss rates, channel efficiency still remains high. The next step is to lay out the properties of Tornado codes in more detail.

In particular, we will consider erasure codes that take a set of k 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 Tornado Codes

In this section, we lay out the theoretical properties of Tornado codes in more detail. We outline how these codes differ from traditional Reed-Solomon erasure codes, and focus in particular on performance comparisons. For the rest of the discussion, we will consider erasure codes that take a set of k packets and produce a set of ℓ redundant packets for a

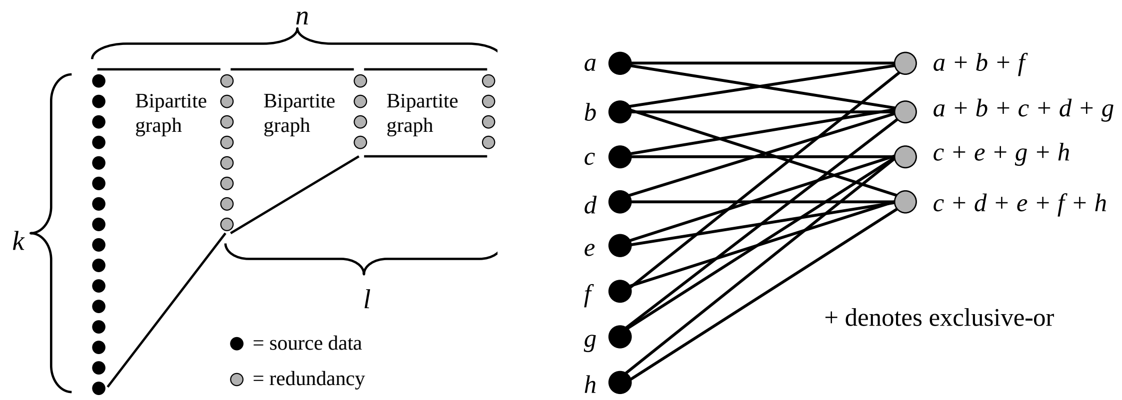

Figure 1: Structure of Tornado Codes

total of $n = k + \ell$ encoding packets all of a fixed length P.

5.1 Theory

We briefly describe the structure of a Tornado code. Tornado codes are based on a series of random bipartite graphs, as depicted on the left side of Figure 1. The leftmost layer stores the original source data. For simplicity, we consider the case in which each node corresponds to one packet, although this need not be the case in practice. Packets in subsequent layers constitute the redundancy. The redundant packets are generated by taking the exclusive-or of the contents of the neighboring packets to the left, as depicted on the right side of Figure 1. The structure of the bipartite random graphs must be specially chosen to guarantee both rapid encoding and decoding and the erasure property described below. A detailed, technical description of these codes is provided in [8] and [9].

Tornado codes have the following erasure property: to reconstruct the source data, it suffices to recover slightly more than k of the n packets stored in the graph. We say the *reception overhead* is ϵ if $(1+\epsilon)k$ encoding packets are required to reconstruct the source data.[2] The advantage of Tornado codes over standard codes is that they trade off a negligible increase in reception overhead for a substantial decrease in encoding and decoding times, thus making their overall efficiency far superior.

Furthermore, the decoding algorithm can detect when it has received enough encoding packets to reconstruct the original file. Thus, the client can run the Tornado decoding algorithm in real-time as the encoding packets arrive, and reconstruct the original file as soon as it determines that sufficiently many packets have arrived. This discussion assumes that the source and the clients have agreed to the graph structure in advance.

Standard erasure codes typically have encoding times proportional to $k(1+\ell)P$ and decoding times proportional to $k(1+x)P$, where x is the number of source data packets not received, and which therefore must be reconstructed from redundant data. Standard erasure codes also require somewhat complex finite field operations, further increasing their encoding and decoding times. As a result, standard erasure codes can only be applied in

[2]Because our codes use random graphs, our codes do not have a fixed reception overhead threshold ϵ. The variation in reception overhead for Tornado codes is detailed in Figure 2.

	Tornado Codes	Reed-Solomon Codes
Reception overhead	$\epsilon > 0$ required	0
Encoding times	$(k+\ell)\ln(1/\epsilon)P$	$k(1+\ell)P$
Decoding times	$(k+\ell)\ln(1/\epsilon)P$	$k(1+x)P$
Basic operation	Simple XOR	Complex field operations

Table 1: Properties of Tornado vs. Reed-Solomon codes

practice when k and ℓ are small. (Values used in [14, 17, 19, 18] have k and ℓ ranging from 8 to 256).

In contrast, Tornado codes have encoding and decoding times that are proportional to $(k+\ell)\ln(1/\epsilon)P$, where ϵ is the reception overhead. The encoding and decoding times of Tornado codes in practice are orders of magnitude faster than Reed-Solomon codes for large values of k and ℓ and for very low reception overhead $\epsilon \approx 0.05$, as we detail in Section 5.2. Moreover, Tornado codes are simple to implement and use only exclusive-or operations, further enhancing their advantage over standard codes.

A summary comparing the properties of Tornado codes and standard Reed-Solomon codes is given in Table 1.

5.2 Practice

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 [4, 14, 16, 17, 18, 19]. The erasure codes listed in Tables 2 and 3 as *Vandermonde* [16] and *Cauchy* [2] are standard implementations of Reed-Solomon erasure codes, based on Vandermonde matrices and Cauchy matrices, respectively. Both *Tornado A* and *Tornado B* codes were designed using some of the principles described in [8] and [9]. 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. 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, for both the Cauchy and the Vandermonde codes, we assume that $k/2$ original file packets and $k/2$ redundant packets were used to recover the original file. This assumption approximately holds when a carousel encoding is used and half the packets are redundant packets (that is, the stretch factor is 2). Tornado A has an average reception overhead of 0.05, so on average $1.05 \cdot k/2$ original file packets and $1.05 \cdot k/2$ redundant packets were used to recover the original file. Tornado B uses a slightly different code structure that is slower to decode but yields a smaller average reception overhead of 0.03.

We note that there is a small variation in the reception overhead for decoding Tornado codes depending on which particular set of encoding packets are received. In Figure 2, we show the percentage of 10,000 trials in which the receiver could not reconstruct the source data for specific percentage overheads for both Tornado A and Tornado B codes.

Encoding Benchmarks				
	Reed-Solomon Codes		Tornado Codes	
SIZE	Vandermonde	Cauchy	Tornado A	Tornado B
250 KB	9.0 seconds	4.6 seconds	0.06 seconds	0.11 seconds
500 KB	39 seconds	19 seconds	0.12 seconds	0.15 seconds
1 MB	150 seconds	93 seconds	0.26 seconds	0.25 seconds
2 MB	623 seconds	442 seconds	0.53 seconds	0.50 seconds
4 MB	not available	1717 seconds	1.06 seconds	0.96 seconds
8 MB	not available	6994 seconds	2.13 seconds	1.72 seconds
16M Bytes	not available	30802 seconds	4.33 seconds	3.23 seconds

Table 2: Comparison of encoding times for erasure codes.

Decoding Benchmarks				
	Reed-Solomon Codes		Tornado Codes	
SIZE	Vandermonde	Cauchy	Tornado A	Tornado B
250 KB	11.0 seconds	2.06 seconds	0.06 seconds	0.88 seconds
500 KB	32 seconds	8.4 seconds	0.09 seconds	1.02 seconds
1 MB	161 seconds	40.5 seconds	0.14 seconds	1.27 seconds
2 MB	1147 seconds	199 seconds	0.19 seconds	1.55 seconds
4 MB	not available	800 seconds	0.40 seconds	2.00 seconds
8 MB	not available	3166 seconds	0.87 seconds	2.90 seconds
16 MB	not available	13629 seconds	1.75 seconds	4.70 seconds

Table 3: Comparison of decoding times for erasure codes.

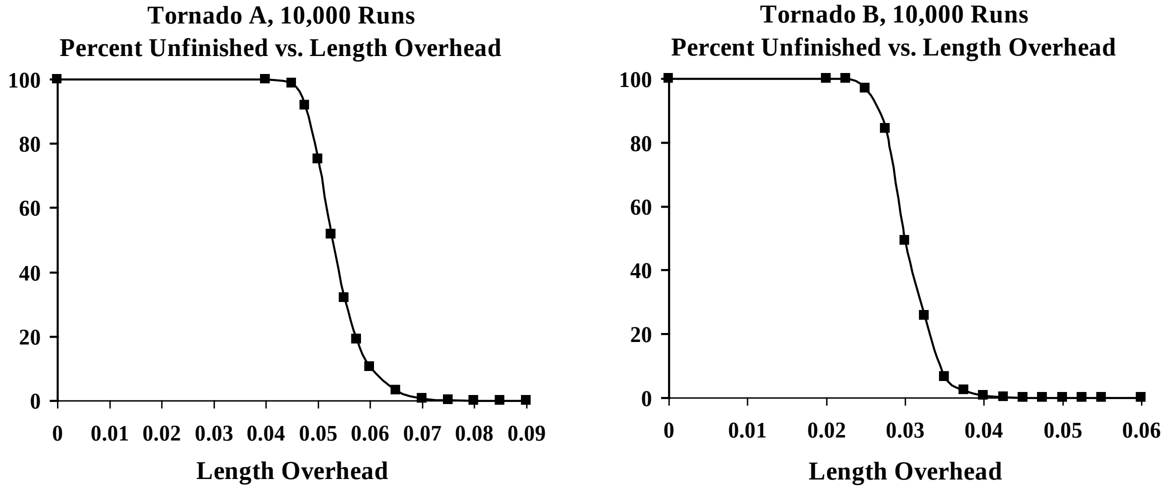

Figure 2: Reception Overhead Variation

For example, using Tornado A codes, after receiving 6% overhead, 90% of the clients could reconstruct the source data and only 10% of the clients needed to receive more packets. For Tornado A, the average overhead was 0.0548, the maximum overhead was 0.0850 and the standard deviation was 0.0052. For Tornado B, the average overhead was 0.0306, the maximum overhead was 0.0550 and the standard deviation was 0.0031.

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 [14, 16, 17, 18]. 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. The smaller the value of k, the faster the decoding algorithms can reconstruct the source data. But as k shrinks, the number of blocks comprising the source data grows. As the number of blocks grows, the receiver will have to wait longer to receive sufficiently many packets to fill each and every block, since the distribution of received packets across blocks is non-uniform. This effect is depicted in Figure 3. Therefore, to compare various protocols, one quantity we measure is the reception efficiency at a receiver, η, defined as:

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

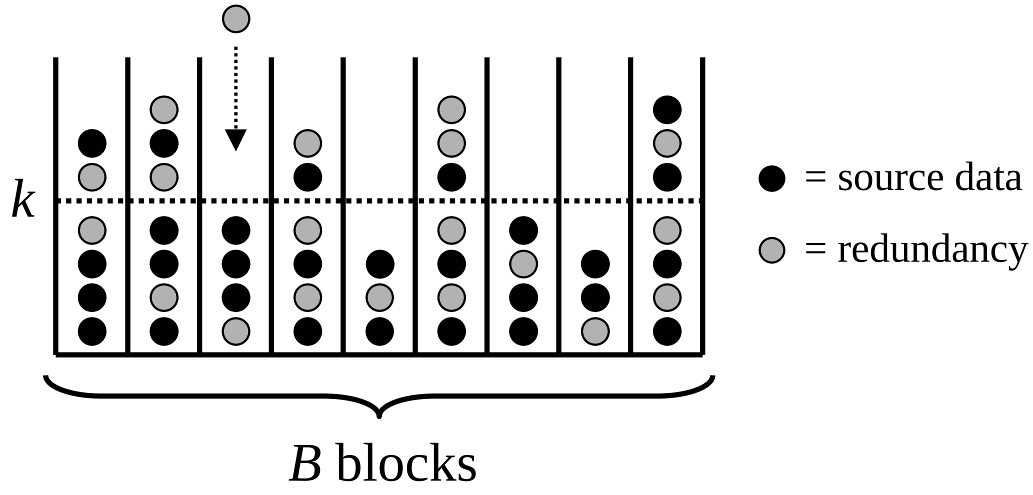

Figure 3: Waiting for the last blocks to fill using interleaved codes ...

Of course the reception overhead and the reception efficiency are related; when the reception overhead is ϵ the reception efficiency is $\frac{1}{1+\epsilon}$. The tradeoff between reception efficiency and coding time for interleaved codes motivates the following set of experiments.

- Suppose we choose k in the interleaved setting so that the reception efficiency is comparable to that of Tornado codes. 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 codes. How does the reception efficiency compare?

In our initial simulations, we assume probabilistic loss patterns in which each transmission to each receiver is lost independently with a fixed probability p. Later in this section, we run on trace data taken from MBone sesssions in which different receivers have different loss rates and the loss patterns are bursty. In our implementation section, we consider additional sources of inefficiency which arise in practice.

6.1 Equating Reception Efficiency

Our first simulation compares the decoding time of Tornado codes with an interleaved code with reception efficiency comparable to those of Tornado codes. In Section 5, we determined experimentally that Tornado A codes have the property that the reception overhead is greater than 0.07 less than 1% of the time. In Table 4, 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 a Tornado code. Of course, this ratio changes as the loss probability and file size change.

We explain how the entries in Table 4 are derived. To compute the running time for interleaved codes, we 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 reception overhead less than 0.07 for less than 1% of the time. (For example, a two

Speedup factor for Tornado codes					
	erasure probabilities				
SIZE	0.01	0.05	0.10	0.20	0.50
250 KB	4.7	11.0	16.7	33.3	33.3
500 KB	6.2	17.8	29.5	44.4	88.9
1 MB	10.3	25.4	37.9	76.1	114
2 MB	16.1	42.1	74.7	112	224
4 MB	18.2	47.3	75.2	128	256
8 MB	17.9	47.9	80.9	138	294
16 MB	20.4	52.4	86.6	151	311

Table 4: Speedup of Tornado A codes over interleaved codes with comparable efficiency.

megabyte file consisting of 2000 one kilobyte packets can be split into at most six blocks while maintaining this property.) 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 stretching factor of two, one half of all packets injected into the system are redundant encoding packets and the other half are source data packets. Therefore, in computing the decoding time per block, we assume that half the packets received are redundant encoding packets. Based on the data previously presented in the Cauchy codes column of Table 3, we approximate the decoding time for a block of k source data packets by $k^2/31250$ seconds. To compute the running time for Tornado codes, we simply use the decode times for Tornado A as given earlier in Table 3.

As an example, suppose the encoding of a 16 MB file is transmitted over a 1 Mbit/second channel with a loss rate of 50%. It takes just over 4 minutes to receive enough packets to decode the file using either a Tornado A or an interleaved code (with the desired reception overhead guarantee), but then the decoding time is almost 10 minutes for the interleaved code compared with under 2 seconds for Tornado A. Comparisons for encoding times yield similar results. We note that even better speedup results can be obtained for large source data files using Tornado B codes due to their stronger decoding guarantee, which interleaved codes are hard-pressed to achieve.

6.2 Equating Decoding Time

Our second simulation establishes interleaved codes which have comparable decoding time to Tornado codes. Although the decoding time is both a function of the block length and the amount of packet loss for Cauchy codes, even small block sizes such as $k = 20$ and $k = 50$ are slower to decode than Tornado codes. (Cauchy codes with $k = 20$ are roughly half as fast as Tornado codes.)

Using these block sizes, we now study the maximum reception overhead 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 a rates of 10% and 50%. The 10% loss rates are representative of congested

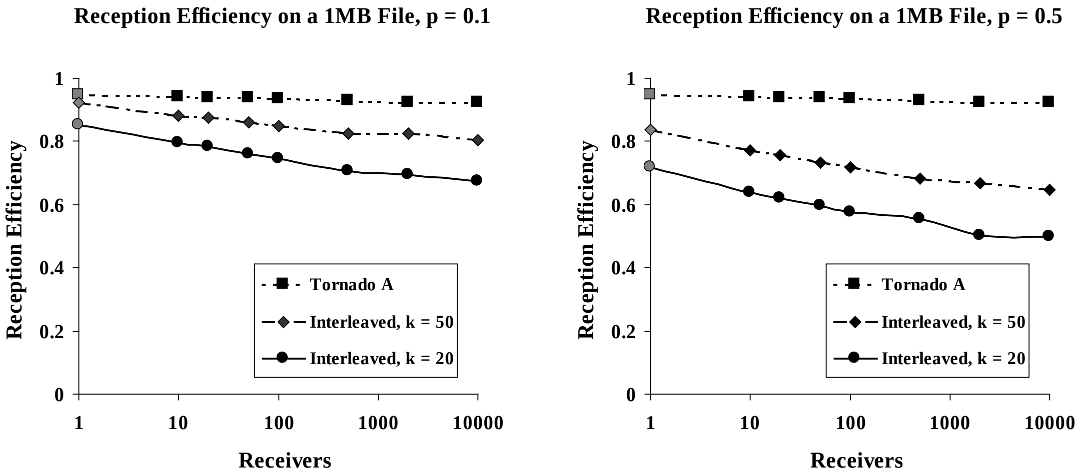

Figure 4: Comparison of reception efficiency for codes with comparable decoding times.

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 [14], use of interleaved codes is probably acceptable, even though Tornado codes perform equally well.

Figure 4 shows the average case and worst case reception efficiency experienced by a set of receivers using interleaved codes with block sizes of 20 and 50, respectively. The average case is represented by the leftmost points in each graph, corresponding to the single receiver case. These results are computed by taking the average of 100 experiments for each receiver set size. In these figures, k refers to the block size (in terms of number of packets) used in the interleaving scheme and p refers to the probability a packet is lost at each receiver.

For packet loss rates of 10% and a block size of $k = 50$, the average efficiency of interleaved codes is comparable to that of Tornado codes. But as packet loss rates increase, or if a smaller block size is used, efficiency of interleaved codes drops off dramatically. Also, efficiency of the worst-case receiver does not scale with interleaved codes as the receiver size grows large. Tornado code efficiency exhibits full scalability and much better tolerance for high loss rates.

6.3 Scaling to Large Files

Our next experiments demonstrate that Tornado codes also scale substantially better than an interleaved approach as the file size grows large. This is due to the fact that the number of packets a client must receive to reconstruct the source data when using interleaving grows super-linearly in the size of the source data. (This is the well-known "coupon collector's problem.") In contrast, the number of packets the receivers require to reconstruct the source data using Tornado codes grows linearly in the size of the source data, and in particular the reception efficiency does not decrease as the file size increases.

The effect of this difference is easily seen in Figure 5. In this case both the average reception overhead and the maximum reception overhead grow with the length of the file

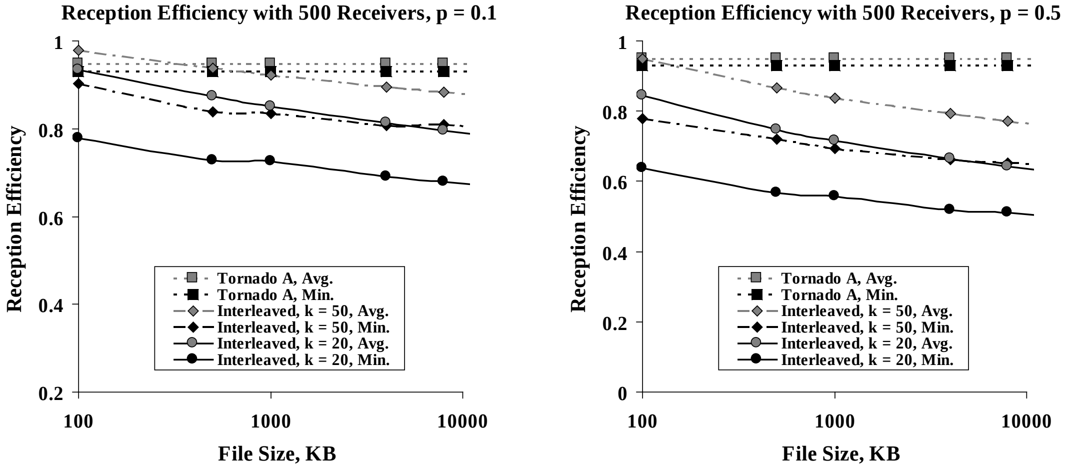

Figure 5: Comparison of reception efficiency as file size grows.

when using the interleaving. This effect is completely avoided by using Tornado codes.

6.4 Trace Data

To study the effects of real loss patterns, we perform a similar comparison using publicly available MBone trace data collected by Yajnik, Kurose, and Towsley [20]. Over a period of several months, clients from across the US and abroad subscribed to MBone broadcasts each of roughly an hour in length and reported which packets they received. Clients experienced loss rates ranging from less than 1% to over 30% over the course of these broadcasts. Sampling from these loss traces, we simulate the process of downloading files of various lengths using interleaving and Tornado codes. The trace sampling consists of choosing a random initial point within each trace for each file size. We plot the average reception efficiency for 120 receivers for various file sizes in Figure 6.

The average loss rate over the parts of the traces considered was approximately 18%. Although this appears high, there is a great deal of variance in the loss rate; some clients experience large bursts of loss rates over significant periods of time. Given this loss rate, it is therefore not surprising that Figure 6 looks similar to the plot in Figure 5 with loss probability $p = 0.1$. Notice that Tornado codes retain their superior reception efficiency even in the face of high rates of bursty packet loss, as we expected.

For the traces we considered, the loss rates were generally below fifty percent, and hence the reception efficiency for the Tornado codes remains extremely high. If the loss rate exceeds fifty percent, however, and we use only a stretch factor of 2, then the reception efficiency necessarily declines. This reduction in efficiency occurs because the carousel cycles, and hence receivers obtain duplicate packets before being able to decode. We examine this phenomenon further with our prototype implementation in the next section.

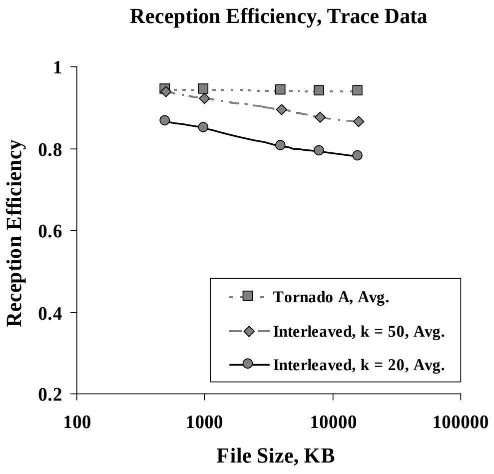

Figure 6: Comparison of reception efficiency for trace data.

7 Implementation of a Reliable Distribution Protocol using Tornado Codes

In this section, we describe an experimental system designed to distribute bulk data to a large number of heterogeneous receivers who may access the data asynchronously. We separate our description of the design into two subsections: the design of the server and the design of the client. Then we describe the experimental setup and performance results of our system.

7.1 Design of the Server

As described in earlier sections, the server must initially choose a stretch factor and generate an encoding of the source data. In our earlier simulations, the server then simply cycled through a random permutation of the source and redundant packets. These simulations ignore the issue of congestion control and do not specify the rate at which packets were injected into the network. Receiver heterogeneity and varying rates of network congestion motivate the need for distribution protocols which can support a variety of end-to-end bandwidth and packet loss rates.

7.1.1 Layering Across Multiple Multicast Groups

The approach we take follows the lead of other authors who advocate *layered* multicast [10, 13, 19]. The main idea underlying this approach is to enable the source to transmit data across multiple multicast groups, thereby allowing the receiver to subscribe to an appropriate subset of these layers. A receiver's subscription level is based on factors such as the width of its bottleneck link to the source and network congestion. The basic ideas common to the proposed layered schemes are:

- The server organizes the data into g layers, numbered $0, \ldots, g-1$, each corresponding to a multicast group. The layers are typically organized in order of 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.

Letting B_i denote the ratio of the rate used at layer i to the rate at the base layer 0, our protocol uses geometrically increasing rates: $B_i = 2^{i-1}$. Thus, a receiver at subscription level i would receive bandwidth proportional to $2B_i$, for $i \geq 1$. The protocol we use is based on the scheme described in [19] which proposes the following two novel ideas, summarized here briefly:

- Congestion control is achieved by the use of *synchronization points* (SP's) which 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 stream is inversely proportional to the bandwidth: lower bandwidth receivers are given more frequent opportunities to move up to higher levels.

- 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 which receivers would experience following an explicit join. So if a receiver feels no congestion during the burst, it can safely increase its level at the next SP. Receivers drop to a lower subscription level in the event of congestion.

Both the sending of SP's and burst periods are driven by the *sender*, with the receivers reacting appropriately. The attractive features of this approach are that receivers do not need to provide congestion control feedback to the source and receivers need not coordinate join attempts to prevent disruption to other receivers. These features are particularly important in the context of a digital fountain in which receiver-to-source and inter-receiver communication are undesirable.

7.1.2 Scheduling Packet Transmissions Across Multiple Multicast Groups

As described earlier, a receiver at level i subscribes to *all* layers 0 through i. Therefore, it is important to schedule packet transmissions carefully across the multiple layers, so as to minimize the number of duplicate packets that a client would receive. The stretch factor c limits the number of distinct packets which can be transmitted, and therefore also has a strong effect on the number of duplicates a client receives, especially in the presence of high packet loss rates. Of course, use of a large stretch factor provides more flexibility, but slows decoding time and increases the space requirements for decoding.[3] For these reasons, we typically choose a stretch factor $c = 2$ as compared to $c = 8$ used in [17, 18]. However,

[3] The memory required for decoding Tornado codes is proportional to the length of the encoding, not to the size of the source data.

Layer	Bandwidth per Round	Packets sent during							
		Rd 1	Rd 2	Rd 3	Rd 4	Rd 5	Rd 6	Rd 7	Rd 8
3	4	0-3	4-7	0-3	4-7	0-3	4-7	0-3	4-7
2	2	4-5	0-1	6-7	2-3	4-5	0-1	6-7	2-3
1	1	6	2	4	0	7	3	5	1
0	1	7	3	5	1	6	2	4	0

Table 5: Packet transmission scheme for 4 layers

we find that this choice is often suitable in practice because we use a packet transmission scheme which has the following property:

One Level Property: If a receiver remains at a fixed subscription level throughout the transmission and packet loss remains sufficiently low (below $\frac{c-1-\epsilon}{c}$, where ϵ is the reception overhead of the Tornado code), it can reconstruct the source data before receiving any duplicate transmissions.

The packet scheduling scheme we use to realize this property is illustrated in Table 5 for 4 layers with source data 4 packets in length and stretch factor 2.

- The source data is stretched into an encoding with stretching factor c, using an appropriate Tornado code.
- The encoding is divided into blocks of size $\sum_{i=0}^{g-1} B_i = B$ packets each. (This division into blocks is *not* to be confused with that used in interleaved schemes - this division is solely for the purpose of determining which packets are sent at which layer at a given time interval; the encoding is done over the entire source data.)
- Within each block, the packets are numbered sequentially from 0 to $B-1$.
- The encoded data is sent in *rounds.* At each round a given layer i will receive a different subset of packets from a block. Within a round, the sending pattern is the same in all blocks (e.g., as shown in Figure 7 for $g = 4$ at round 4). This pattern is obtained using reverse binary encoding of $g-1$ bit numbers, the pattern at round j is determined as follows: Let $j' = j \bmod 2^{g-1}$ and let $\mathrm{bit}_p(j')$ denote the p^{th} least significant bit of j' (with $\mathrm{bit}_0(j')$ denoting the least significant bit). Then, for layer $g-1$, at round j, the packets sent within a block are those numbered (in $g-1$ bit binary numbers):
$$\mathrm{bit}_0(j')0\ldots0 \ \text{ to } \ \mathrm{bit}_0(j')1\ldots1 \ \text{ inclusive}$$
For layer $g-2$ similarly, at round j, the packets sent are
$$\overline{\mathrm{bit}_0(j')}\ \mathrm{bit}_1(j')0\ldots0 \ \text{ to } \ \overline{\mathrm{bit}_0(j')}\ \mathrm{bit}_1(j')1\ldots1 \ \text{ inclusive}$$
and so on for the lower layers.

It is not hard to prove that this sending pattern satisfies the One Level Property. In fact, we note that the sender transmits a permutation of the entire encoding both to each multicast layer and to each cumulative subscription level before repeating a packet. However, in

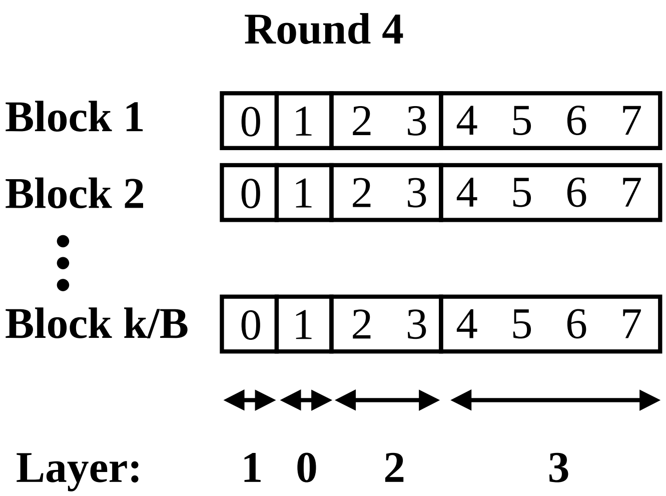

Figure 7: Packet send pattern for $g = 4$ at round 4

practice, receivers which change their subscription level over time do not witness this ideal behavior. While we show in Section 7.3 that reception efficiency remains high even when receiver subscription levels change frequently, optimizing properties of the schedule further for this scenario is one of the important and interesting questions which remain open in our approach.

7.2 Design of the Client

As detailed in the previous subsection, the client is responsible for observing SPs and modifying its subscription level as congestion warrants. 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. The second is a statistical approach in which the client waits until a fixed number of packets arrive from which it is likely that the source can be reconstructed. If the quantity of packets is insufficient, it acquires more packets from the source stream. While the incremental approach has the benefit of enabling some decoding computation to be overlapped with packet reception, we found the statistical approach to be simpler and sufficiently fast in practice. Therefore, we chose this statistical approach for our final implementation.

7.3 Experimental Setup and Results

Now we turn to measurements of reception efficiency using our experimental system. Recall that the reception efficiency of a receiver is defined as

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

The reception efficiency can be further separated into two components:

1. The *coding efficiency*, η_c, which captures the loss in efficiency due to the use of Tornado codes, and is defined as:

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

2. The *distinctness efficiency*, η_d, which captures the loss in efficiency due to receipt of duplicate packets, usually caused by changes in receiver subscription layers described in Section 7.1.2. It is defined as:

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

Thus, $\eta = \eta_c \eta_d$.

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 efficiency 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 A to produce the encoding. The server runs two threads: a UDP unicast thread which provides various control information such as multicast group information and file length to the client and a multicast transmission thread. The clients for both protocols 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 clients 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* [6]) 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. Base layer bandwidth was set at rates ranging from 64 Kb/sec to 512 Kb/sec. We ran experiments with the server both at Berkeley and at CMU and with the clients 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 CMU and Berkeley were 167 MHz UltraSPARC-1's running Solaris 2.5.1. The client at Cornell ran on a 60 MHz Sparc. When running the layered protocol, we used 4 layers.

The data from the two sets of experiments are shown in Figure 8. As seen from the graphs for the single layered case, for packet losses of less than 50% , the distinctness efficiency is almost always 100%. This is to be expected because of the One Level Property.[4] Thus, for low loss rates, the protocol efficiency is effectively the decoding efficiency, which in

[4] Note that it is possible to have a *cumulative* loss rate which is less than 50% but in which losses initially are higher than 50% in the first cycle so that the client receives some duplicates. This is precisely what happened for the outlying point at 35% packet loss in the single layer distinctness efficiency graph.

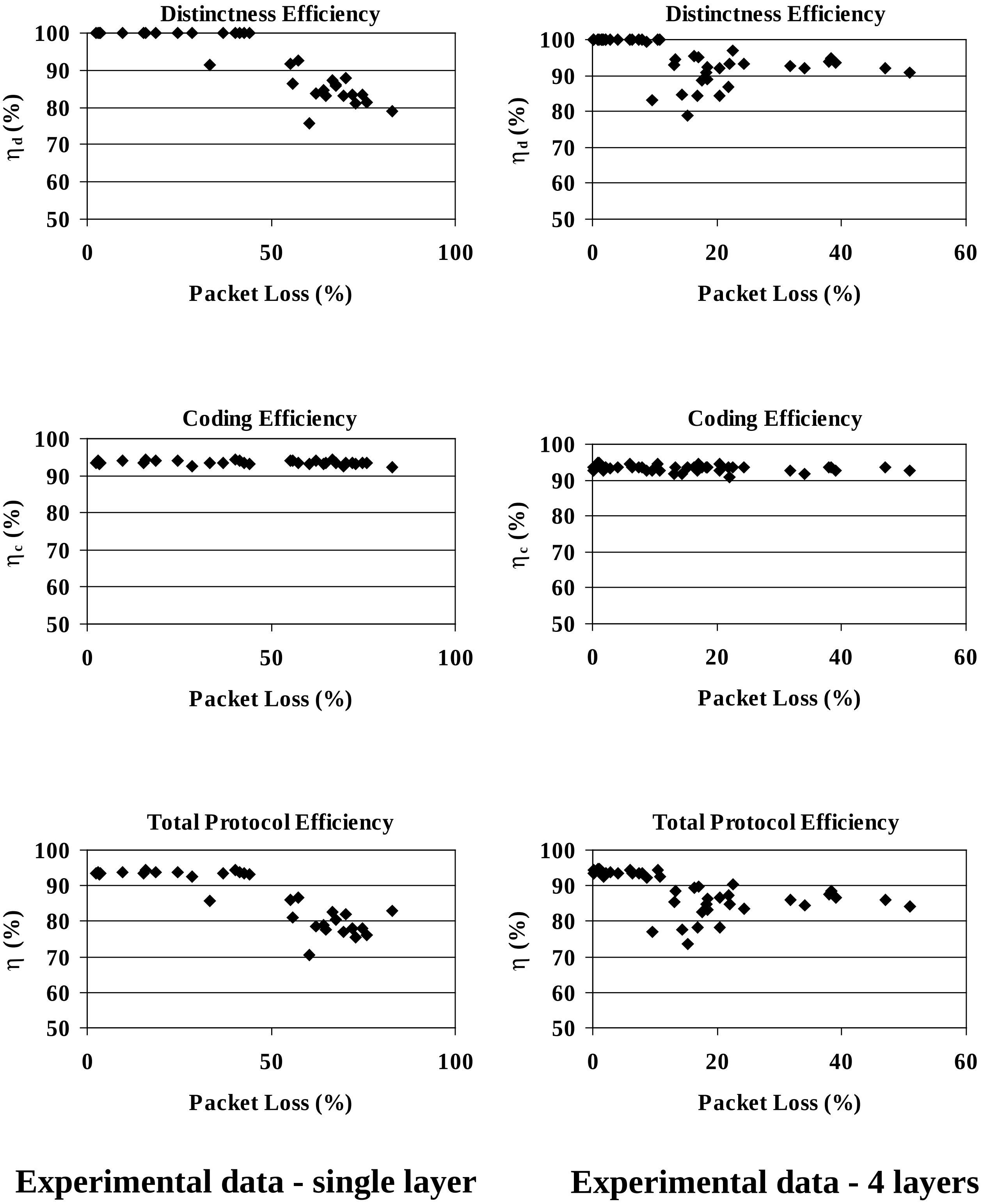

Figure 8: Experimental Results of the Prototype

our example was roughly 94% on average. We further observe that the transmission scheme is robust even under severe loss rates - at nearly a 70% loss rate, the reception efficiency is over 75%.

The graphs on the right hand side of Figure 8 shows experimental data for the multilayered case. We observe that the use of multiple layers for congestion control has a deleterious effect on distinctness efficiency. In our set of experiments, the efficiency fell below 100% at loss rates of as low as 13%. This is not unexpected since switching between subscription levels can cause the client to receive packets that had already been obtained at previous subscription levels. We note however, that most of our experimental runs had over 80% total reception efficiency even with loss rates exceeding 30%. Distinctness efficiency can be further reduced using larger stretch factors. 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. A related, challenging question would be determining packet transmission schedules which maximize the reception efficiency for receivers which change levels frequently.

8 Conclusion

The introduction of Tornado 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 can yield a much closer approximation to a digital fountain than previous systems based on standard Reed-Solomon erasure codes. Our prototype multicast data distribution system demonstrates that simple protocols using Tornado codes are effective in practice. We plan to extend and improve our prototype system to further explore the problem of multicast distribution. In particular, we plan to test our prototype system with large numbers of users to fully demonstrate its effectiveness.

Given that we can closely approximate a digital fountain with Tornado codes, we conclude with other possible applications for such an encoding scheme. 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 which 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 information dispersal work of Rabin [15] who suggested using standard erasure codes. We expect Tornado codes will lead to improved practical dispersity routing schemes.

Another application for which the Tornado code approximation might be useful arises in the context of mirrored data. Currently, to minimize response time, clients search for a lightly loaded server on an uncongested path. If the sources use ideal digital fountains to transmit the data, clients can access multiple sources simultaneously, and aggregate *all* the packets they receive to recover the data efficiently. The problem with a Tornado code solution is that if the stretch factor is small, one receives duplicate packets frequently; if the stretch factor is large, the space and time requirements for decoding become prohibitive. We expect that in some situations, parameters may be set appropriately to yield a viable solution.

References

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

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

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

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

[5] 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.

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

[7] J. C. Lin, S. Paul, "RMTP: A Reliable Multicast Transport Protocol." In *IEEE INFOCOM '96*, pp. 1414-1424, March 1996.

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

[9] M. Luby, M. Mitzenmacher, 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.

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

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

[12] 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.

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

[14] J. Nonnenmacher, E. W. Biersack, and D. Towsley, "Parity-Based Loss Recovery for Reliable Multicast Transmission." In *Proc. of ACM SIGCOMM '97*, 1997.

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

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

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

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

[19] L. Vicisano, L. Rizzo, and J. Crowcroft. "TCP-like congestion control for layered multicast data transfer." To appear in *INFOCOM '98*.

[20] 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.

[21] 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.