Accessing Multiple Mirror Sites in Parallel: Using Tornado Codes to Speed Up Downloads

John W. Byers
Dept. of Computer Science
Boston University
Boston, Massachusetts

Michael Luby
Digital Fountain, Inc.
Oakland, California

Michael Mitzenmacher
Compaq Systems Research Center
Palo Alto, California

Abstract— **Mirror sites enable client requests to be serviced by any of a number of servers, reducing load at individual servers and dispersing network load. Typically, a client requests service from a single mirror site. We consider enabling a client to access a file from multiple mirror sites in parallel to speed up the download. To eliminate complex client-server negotiations that a straightforward implementation of this approach would require, we develop a feedback-free protocol based on erasure codes. We demonstrate that a protocol using fast Tornado codes can deliver dramatic speedups at the expense of transmitting a moderate number of additional packets into the network. Our scalable solution extends naturally to allow multiple clients to access data from multiple mirror sites simultaneously. Our approach applies naturally to wireless networks and satellite networks as well.**

I. Introduction

Downloading a large file from a heavily loaded server or through a highly congested link can be a painfully slow experience. Similarly, clients who connect to the Internet at modem speeds often suffer through interminable waiting periods to download web pages with graphical content. The many proposed solutions for addressing these problems share a common theme: improve performance at the bottleneck.

For modem users, there is not much that can be done; to improve downloading time they must either upgrade to higher baud rates or settle for receiving distilled, lower bandwidth versions of the content they wish to access. But for the rest of us, for whom the last mile is not the bottleneck, there are a wide variety of techniques to improve performance in the network and at the server. The most relevant to our discussion is the use of *mirror sites*.

The mirroring approach deploys multiple servers storing the same data at geographically distributed locations, in an effort to both distribute the load of requests across servers and to make network connections shorter in length, thereby reducing network traffic. A limitation of current mirroring technology is that the user must choose a single mirror site from which to access data. While the choice of server may appear obvious when the number of mirror sites is small, the work of [25] (and others) indicates that the obvious choice is not always the best choice and dramatic performance improvements can result from more careful selection. This selection process can be automated and improved by statistical record-keeping [25], or by dynamic probing [7], [18].

Our first objective is to enable users to download data from multiple mirror sites in parallel in order to reduce downloading time. This technique not only has the potential to improve performance substantially over a single server approach, but can eliminate the need for a complex selection process.

For some applications, such as software distribution [9], [24], mirrored data may be requested by a vast number of autonomous clients whose access intervals may overlap with one another. In such a situation, rather than having each client establish a set of point-to-point connections to multiple mirror sites in parallel, we envision a more general system by which mirror sites establish multicast groups to transmit data. Provided that the transmissions of these multicast groups are orchestrated so as to keep the transmission of duplicates to a minimum, a client could subscribe to several multicast groups in parallel to retrieve the data more quickly. The use of multicast provides scalability to an unlimited number of clients, each of which may be subscribing to different subsets of the multicast groups.

As an example, such a system could be used by employees of an international conglomerate to download daily financial statements via their corporate intranet. Each of a number of geographically distributed mirror sites would establish a multicast session to distribute the information. Transparent access from thousands of clients to multiple mirror sites would be an effective mechanism to speed up download times and to balance load, especially away from heavily accessed servers at peak usage times. Moreover, such a scheme would be simple to automate.

Of course, it may not be obvious how to take this conceptually simple approach and derive a simple and workable implementation around it that delivers superior performance. For this step, we utilize erasure codes (often referred to as Forward Error Correction codes) which have been widely suggested as a means of simplifying and streamlining reliable multicast transmission [6], [9], [10], [20], [22], [23], [24], [26]. The main idea underlying this technique [14], [21] is to take an initial file consisting of k packets and generate an n packet encoding of the file with the property that the initial file can be restituted from *any* k packet subset of the encoding. For the application of reliable multicast, the source transmits packets from this encoding, and the encoding property ensures that different receivers can recover from different sets of lost packets, provided they receive a sufficiently large subset of the transmission. To enable parallel access to multiple mirror sites, the sources each transmit packets from the same encoding, and the encoding property ensures that a receiver can recover the data once they receive a sufficiently large subset of the transmitted packets, regardless

J. Byers and M. Luby were supported in part by National Science Foundation operating grant NCR-9416101 and CCR-980045; they did much of this work while at the International Computer Science Institute.

of which server the packets came from. In fact, the benefits and costs of using erasure codes for parallel access to multiple mirror sites are analogous to the benefits and costs of using erasure codes for reliable multicast. For both applications, simple schemes which do not use encoding have substantial drawbacks in terms of complexity, scalability, and in their ability to handle heterogeneity among both senders and receivers. Our approach eliminates many of these drawbacks.

Ideally, using erasure codes, a receiver could gather an encoded file in parallel from multiple sources, and as soon as *any* k packets arrive from *any* combination of the sources, the original file could be restituted efficiently. In practice, however, designing a system with this ideal property and with very fast encoding and decoding times appears difficult. Hence, although other erasure codes could be used in this setting, we suggest that a newly developed class of erasure codes called *Tornado codes* are best suited to this application, as they have extremely fast encoding and decoding algorithms [12]. Indeed, previously these codes have been shown to be more effective than standard erasure codes in the setting of reliable multicast transmission of large files [6]. The price for the fast encoding and decoding is that slightly more than k distinct packets are required to reconstitute the message. In this paper, we will use Tornado codes as the basis for our proposed approach. We include a brief description of the properties of Tornado codes and contrast them to standard Reed-Solomon codes in this paper in order to keep this work self-contained.

Our general approach to accessing servers in parallel is applicable in many other situations. For example, wireless clients could access data through multiple wired access points in parallel, potentially significantly speeding up downloads. Similarly, clients on satellite networks could obtain information from multiple satellites simultaneously. Because our solutions require little or no feedback from the receiver to the source, they are quite appealing for such networks. Moreover, because we generate encoded versions of the files, our protocols are highly robust against packet loss, including the high degree of packet loss often found in wireless networks. As a third example, clients with access to multiple communication media, such as simultaneous access on both an ISDN line and a modem line, can easily employ our techniques.

After describing the conceptual basis for our work, the main focus of the paper is an analysis of the costs and benefits of parallel access to multiple mirror sites as compared to point-to-point connections. The primary benefit is the speedup of the download. The main cost of our approach is bandwidth, in that a moderate number of additional packets may be injected into the network by the mirror sites.

An important aspect of our work is that no attempt is made to circumvent congestion control mechanisms in any way. In fact, we anticipate that deployment of our techniques would be used in conjunction with TCP-friendly regulatory mechanisms. That is, even though our general solutions could be used in situations where no feedback from receiver to sender is possible, they can also be easily implemented on existing network infrastructure using TCP as well.

A. *Related Work*

Although the idea of parallel access to multiple mirror sites is quite straightforward, it has apparently received little attention in the literature. The most relevant related work includes Maxemchuk's work on dispersity routing [14], [15] and Rabin's work on information dispersal (IDA) [21]. For example, the idea of Rabin's IDA scheme is to break a file into several pieces, where each piece also includes some redundant information. By dispersing these pieces among different nodes in the network, one guarantees fault tolerance against link or node failures. Similarly, Maxemchuk suggests that one may send the encoded pieces of a file along different routes of a network; when enough pieces arrive, the file may be reconstructed.

Although parallel access to multiple mirror sites (many-to-one transmission) is not a well studied concept, multicast distribution (or one-to-many distribution) has been widely studied. In particular, as we have mentioned, the idea of using erasure codes to simplify multicast transmission has recently been considered in many works [6], [9], [10], [20], [22], [23], [24], [26]. In the database and mobile computing literature, there has also been similar use of erasure codes in work on broadcast disks [3], [4]. A primary benefit of using erasure codes in these contexts is that it reduces or potentially eliminates the need for feedback from the receivers to the sender. For both applications, receiver-to-source feedback can often be expensive, both because the networks may be highly asymmetric (such as in satellite-based networks) and because the system needs to scale to many receivers.

Previous multicast work is also relevant since it demonstrates another approach for removing system bottlenecks by reducing the load on a single server. For example, Almeroth, Ammar, and Fei suggest that servers holding heavily accessed web pages can reduce load and improve delivery performance by servicing requests in batches, rather than with separate point-to-point connections [2]. Their work advocates queuing popular requests at the server and establishing a multicast session over which to service the request once the queue length reaches a threshold. For pages which are in continuous demand, the use of reliable multicast with erasure codes may be an even better approach.

Our work contrasts with previous work in several regards. In particular, our focus on speed, rather than fault tolerance, highlights different design aspects of effective solutions. Of course, fault tolerance remains a benefit of our approach. Second, our consideration of the combination of multicast and parallel access, or many-to-many distribution, also appears novel. Using a conceptual approach from our previous work on multicast [6], we demonstrate a simple, efficient solution to this distribution problem. A third important difference is that virtually all previous work has utilized standard erasure codes, such as Reed-Solomon codes. These codes generally take time quadratic in the length of the original message to encode and decode, making them unsuitable for many practical applications. For example, encoding and decoding an entire file could take orders of magnitude more time than the download. By making use of Tornado codes, a new class of erasure codes developed in [12], [13], we can develop practical schemes that employ encoding. Finally, because we expect our approach to be useful in various domains (including mobile wireless networks, satellite networks, and the Internet) we develop a general approach applicable to a variety of media and give detailed consideration to the tradeoff between feedback and bandwidth.

B. Assumptions

There are several assumptions that we make in the remainder of the paper, which we describe in detail here. It is required that each mirror site can efficiently produce a pool of encoded packets for the source data, either in advance, or on demand. Furthermore, encoding parameters such as file size, packet size and encoding length must either be established in advance, or must be easily agreed upon between client and servers.

In order for a client to derive benefit from accessing an additional mirror site in parallel, there must be residual network bandwidth available from the client to that site. Intuitively, if this is not the case, then accessing that additional site will induce congestion, and that new connection will interfere with the performance of existing connections. Therefore, we require that the set of paths from client to mirror sites are *bottleneck-disjoint*: packets from one mirror site are neither slowed nor dropped because of packets from a different mirror site sent to the same receiver.

We recognize that a drawback of our approach is that in order to make use of our protocols, receivers must necessarily have the capability to open multiple connections simultaneously along bottleneck-disjoint paths (or approximately bottleneck-disjoint) paths. For many clients, such as modem users, our approach is not likely to be beneficial. We suggest, however, that for people or companies with large available incoming bandwidth, the potential increase in speed from our approach can outweigh the potential cost of establishing multiple connections. Also, given the subsequent analysis in this paper, we expect substantial speed gains even in situations where the assumption about bottleneck-disjoint paths is only approximately true. Hence, in this paper we concentrate on the feasibility of simple protocols given this assumption, a subject which is interesting in its own right. We leave the question of how best to establish and maintain bottleneck-disjoint paths in various types of networks as a question for future research.

II. Inadequacy of a basic scheme

We first describe a simple solution that works over TCP-style connections without any source encoding. We use the term *file* generically to represent an intact item of source data to download. Consider a receiver which wants to download a file from two mirrored sites, A and B. The receiver, upon requesting the file, specifies a *packet set* for both A and B, denoting the packets they are supposed to send. For example, the receiver may initially specify that A should send the first half of the packets while B should send the second half. Ideally, the receiver would obtain half the packets in the file from each of A and B, recovering the entire file in half the time.[1]

In practice, variability and heterogeneity in end-to-end transmission rates, packet loss rates, and latency between the receiver and the various sites requires a more complex control structure. For example, if A can send packets at twice the rate of B, then A will complete transmission of its designated packet set well before B. This problem can be handled by having the the receiver *renegotiate* with the senders. For example, the receiver could ask A to continue sending some of the packets not yet received and simultaneously limit the packets to be sent by B. By using the previous sending rate of A and B as a guide, the receiver can attempt to balance the size of the new packet sets for A and B appropriately. Note that the receiver may even allow the packet sets for A and B to overlap. This increases the likelihood of receiving all the necessary packets quickly at the expense of possibly receiving duplicate packets and thereby wasting bandwidth.

Alternatively, the file could initially be divided into several fixed size blocks. Suppose, for example, the receiver instructs the senders to divide the file into ten equally sized disjoint blocks. Then when a sender finishes sending a block, the renegotiation stage would consist of the receiver requesting the sender to send a block that has not yet been received from any sender.

While these approaches are conceptually feasible, they introduce a number of technical and logistical problems that make them problematic in practice. For example, if transmission rates vary significantly over time, several renegotiations may be required to obtain the file at the fastest rate. Packet loss also introduces potential problems: how should the contract be renegotiated if certain packets from a packet set have not yet arrived? Reliable retransmission-based protocols such as TCP handle renegotiations at the granularity of a packet, but in this general setting, there is a tradeoff between the overhead associated with fine-grained renegotiation granularity and download time. These problems grow more significant with the number of senders a receiver uses, suggesting that this approach, while implementable, may not scale well to many senders. Moreover, this solution appears completely inadequate for many-to-many distribution.

III. A digital fountain solution

In this section, we outline an idealized solution to the downloading from multiple mirrors problem, based on a similar approach to reliable multicast presented in [6]. A client wishes to obtain a file consisting of k packets from a collection of mirrored servers. In an idealized solution, each server sends out a stream of distinct packets, called encoding packets, which constitute an encoding of the file. A client accepts encoding packets from all servers in the collection until it obtains exactly k packets. In this idealized solution, the file can then be reconstructed regardless of which k encoding packets the client obtains. Therefore, once the client receives any k encoding packets, it can then disconnect from the servers. We assume that this solution requires negligible processing time by the servers to produce the encoding packets packets and by the client while merging data received from the various servers and recovering the source data from the k encoding packets. A consequence of this approach is that there is no need for feedback from the receiver to the senders aside from initiating and terminating the connections; in particular, no renegotiation is ever necessary.

We metaphorically describe the stream of encoding packets

[1] In the special case of only two senders, another simple solution is to "burn the candle at both ends": have one sender start sending packets from the beginning of the file, and have the other start from the end. As this solution does not naturally generalize beyond two senders, we do not consider it further here.

produced by a server in this idealized solution as a *digital fountain*, as we did in a similar approach to reliable bulk transfers presented in [6]. 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.

IV. Building a digital fountain with erasure codes

Erasure codes can provide close approximations to the ideal properties of a digital fountain. Given input parameters k and n, erasure codes are typically designed to take a set of k packets and produce a set of ℓ redundant packets for a total of $n = ck = k + \ell$ encoding packets all of a fixed length P. We will subsequently refer to the ratio $c = n/k$ as the *stretch factor* of an erasure code.

Theoretically, it is possible to take a source file, choose a very large stretch factor and divide the encoding packets among the senders in such a way so that no duplicates are sent. In this case, the probability of failure due to packet loss could be made vanishingly small. However, using such a large stretch factor can dramatically slow both encoding and decoding. In fact, the encoding and decoding processing times for standard Reed-Solomon erasure codes are prohibitive even for moderate values of k and n. Thus, in order to efficiently encode and decode large files with moderate stretch factors, we use the recently developed Tornado codes [12]. Of course there is an associated cost for this solution, in that smaller stretch factors increase the likelihood of duplicates and thus may require additional bandwidth. One of our main findings, which we demonstrate with simulation results, is that the overhead in terms of extra bandwidth used appears quite reasonable considering the potential speed gains.

In [6], we provide a detailed comparative analysis between Reed-Solomon and Tornado code implementations of digital fountains. In this paper, we will use the latter implementation, and refer the reader to [6] for our justification of this choice. To keep the paper self-contained, we briefly describe the relevant properties and the performance of Tornado erasure codes. A more detailed, technical description of Tornado codes and their theoretical properties is provided in [12] and [13].

A. Tornado Code Overview

As with standard erasure codes, Tornado codes produce an n packet encoding from a k packet source. However, Tornado codes relax the decoding guarantee as follows: to reconstruct the source data, it is necessary to recover ϵk of the n encoding packets, where $\epsilon > 1$. We then say that ϵ is the *decoding inefficiency*.[2] The advantage of Tornado codes over standard codes (which do not relax the decoding guarantee and have $\epsilon = 1$) is that Tornado codes trade off a small increase in decoding inefficiency for a substantial decrease in encoding and decoding times, thus making them far more attractive for large values of n and k. As with standard codes, the Tornado decoding algorithm can detect when it has received enough encoding packets to reconstruct the original file. Thus, a client can perform decoding in real-time as the encoding packets arrive, and reconstruct the original file as soon as it determines that sufficiently many packets have arrived.

[2]Because our codes are constructed using randomization, a given code does not have a fixed decoding inefficiency threshold ϵ; however, the variance in decoding inefficiency is generally very small. For more details see [6].

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

TABLE I
PROPERTIES OF TORNADO VS. REED-SOLOMON CODES

Decoding Benchmarks	
SIZE	Tornado Z
250 KB	0.18 seconds
500 KB	0.24 seconds
1 MB	0.31 seconds
2 MB	0.44 seconds
4 MB	0.74 seconds
8 MB	1.28 seconds
16 MB	2.27 seconds

TABLE II
TORNADO DECODING TIMES.

While fast implementations of standard codes have quadratic encoding and decoding times for this application, Tornado codes have encoding and decoding times proportional to $(k + \ell)\ln(1/(\epsilon - 1))P$, where ϵ is the decoding inefficiency. Moreover, Tornado codes are simple to implement and use only exclusive-or operations. A comparison between the properties of Tornado codes and standard codes is given in Table I.

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. Decoding times for a specific code called Tornado Z on a Sun 167 MHz UltraSPARC 1 with 64 megabytes of RAM are provided in Table II. (Encoding times and further simulation results for this code are available in [6].) This code was used with packet length $P = 1$KB and stretched files consisting of k packets into $n = 2k$ encoding packets, i.e., the stretch factor is 2. For the decoding, we assume that $1/2$ the packets received are original file packets and $1/2$ are redundant packets when recovering the original file. In 10,000 trials the average decoding inefficiency for Tornado Z was 1.0536, the maximum inefficiency was 1.10, and the standard deviation was 0.0073. As the variance in the decoding inefficiency for Tornado Z is very small, we will frequently consider only the average decoding inefficiency in the rest of this paper, which we overestimate slightly as $\epsilon = 1.055$.

We note that one can trade off speed against inefficiency in designing Tornado codes. In particular, slower codes with reduced decoding inefficiency can be constructed; see [12], [13]. Throughout this paper, we will adopt Tornado Z as a suitably representative example.

V. TORNADO CODE SOLUTIONS

A. Feedback-free Tornado solution

We begin with an approach in which there is no feedback from the receiver to the senders, except for connection set-up and connection teardown. A feedback-free scheme such as this is appropriate when communicating over asymmetric channels in which sending control information from the receiver to the senders is expensive. This is often the case in satellite-based networks, in which bandwidth on the back channel is extremely limited. Minimizing the amount of feedback from receivers to a source can also be important in avoiding feedback implosion for multicast transmissions. Even the simple *no feedback* protocol which we develop under these constraints has substantial advantages over a single access approach.

In this protocol, each sender encodes the file using a Tornado code, and randomly permutes the packets comprising the encoding before sending. Note that each sender uses a different permutation to determine the order in which to send out packets, but the packets that make up the encoding are the same for all senders. A client collects packets from multiple senders in parallel until sufficiently many packets arrive to restitute the file. In rare cases (caused by extremely high packet loss rates), a sender may send out all of its packets before the receiver obtains enough packets to decode; in this case the sender continues to send by cycling through the packets in the same order. The receiver may receive duplicate packets, most commonly when identical copies of the same packet arrive from more than one sender, or more rarely in the event that a sender cycles through the encoding. We will generally assume in the analysis in this section that the stretch factor is chosen sufficiently large so that cycling does not occur.

Our performance measure is the *reception inefficiency*, which we define to be η if the receiver receives a total of ηk packets from the servers before it can reconstruct the file. The reception efficiency comprises two separate sources of inefficiency: the decoding inefficiency of Tornado codes outlined in the previous section, and the *distinctness inefficiency* due to the arrival of duplicate packets. We define the *distinctness inefficiency* to be δ where δ is the average number of copies of each packet received by the receiver prior to reconstruction. The reception inefficiency η is then just the product of the decoding inefficiency ϵ of the Tornado code and the distinctness inefficiency δ; that is, $\eta = \delta\epsilon$.

While the decoding efficiency is approximately a fixed quantity (a function only of the encoding), the distinctness inefficiency varies with the number of senders and the stretch factor. Increasing the number of senders increases the likelihood of receiving duplicates, while increasing the stretch factor increases the number of distinct packets, and thus decreases the likelihood of receiving duplicates. We also note that the number of duplicate packets is maximized when the receiver obtains packets from each sender at the same rate, as can be shown using a symmetry-based argument.

A.1 Mathematical models

One can write explicit equations to determine the distribution of the total number of packets that must be received before decoding, and in turn the distribution of the reception inefficiency. For example, consider a file with k packets initially, encoded into a form with $n = ck$ packets, where c is the stretch factor. Suppose that there are two senders, packets arrive from the senders at an equal rate, and that the receiver must receive $m = \epsilon k$ distinct packets in order to reconstruct the file, where ϵ is the decoding inefficiency. (For convenience here we ignore the variance in the decoding inefficiency and treat it as a constant; the equations can be modified accordingly.) Let ℓ be the number of packets sent by each sender. We assume that the stretch factor is large enough so that a sender does not send the same packet more than once. Let i represent the number of packets from the second sender that are distinct from those sent by the first sender. The probability that 2ℓ received packets include at least m distinct packets is

$$\sum_{i \le \ell : i+\ell \ge m} \left(\frac{1}{\binom{n}{l}}\right)^2 \binom{n}{\ell}\binom{\ell}{\ell - i}\binom{n-\ell}{i}.$$

This expression can be written more succinctly using multinomial coefficients:

$$\sum_{i \le \ell : i+\ell \ge m} \left(\frac{1}{\binom{n}{l}}\right)^2 \binom{n}{\ell - i; i; i}$$

Similarly, expressions for situations with more receivers and variable arrival rates can be determined, although they are not simple to evaluate. These complex expressions completely describe the distribution of distinctness inefficiency, but simpler calculations can give us an indication of *expected* performance. That simpler calculation involves successively calculating the number of expected non-duplicate packets from each sender. That is, suppose we have senders $S_1, S_2, \ldots, S_k$. Let the ratio of the number of packets received from S_i to the number received from S_1 be r_i. Let z_i be the expected number of distinct packets received from the first i receivers. Consider the $(i+1)st$ sender. For each of the $r_{i+1}z_1$ packets this sender sends, the probability that the packet is distinct from the z_i obtained from the first i senders is approximately $\left(1 - \frac{z_i}{n}\right)$. Note that this is only an approximation, as we have assumed that the number of distinct packets actually received from the first i senders equals its expected value. This assumption holds true exactly only for the first sender. Hence, we have the following recursion:

$$z_{i+1} \approx z_i + (r_{i+1}z_1)\left(1 - \frac{z_i}{n}\right). \quad (1)$$

As an example of how to use equation (1), suppose there are two senders that send at equal rates (with no losses). We wish to find the number of packets z that must be sent by each sender so that m distinct packets arrive at the receiver. Then we wish z_2 to be m, so (using equation (1) as an equality)

$$\begin{aligned} m &= z + z\left(1 - \frac{z}{n}\right), \\ \frac{z^2}{n} - 2z + m &= 0. \end{aligned}$$

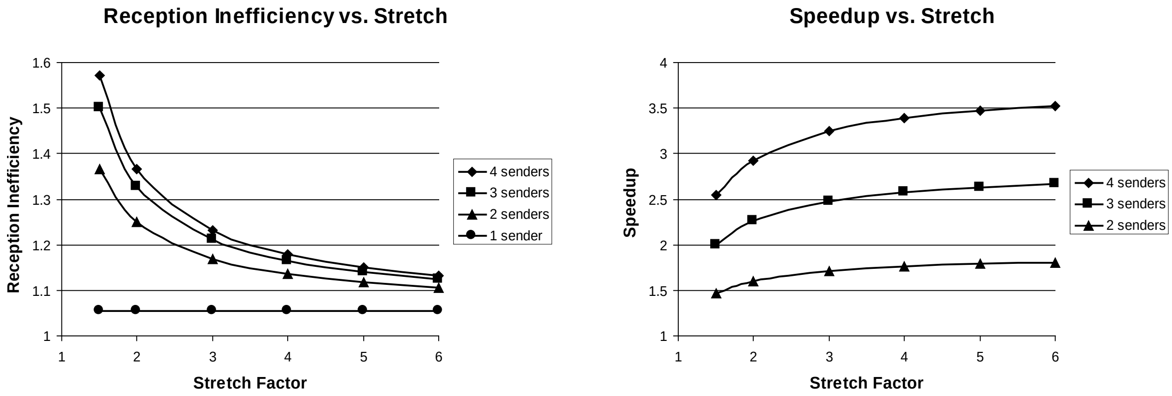

Fig. 1. Reception Efficiency and Speedup vs. Stretch Factor, packets arrive at an equal rate from each sender, average of 1000 trials

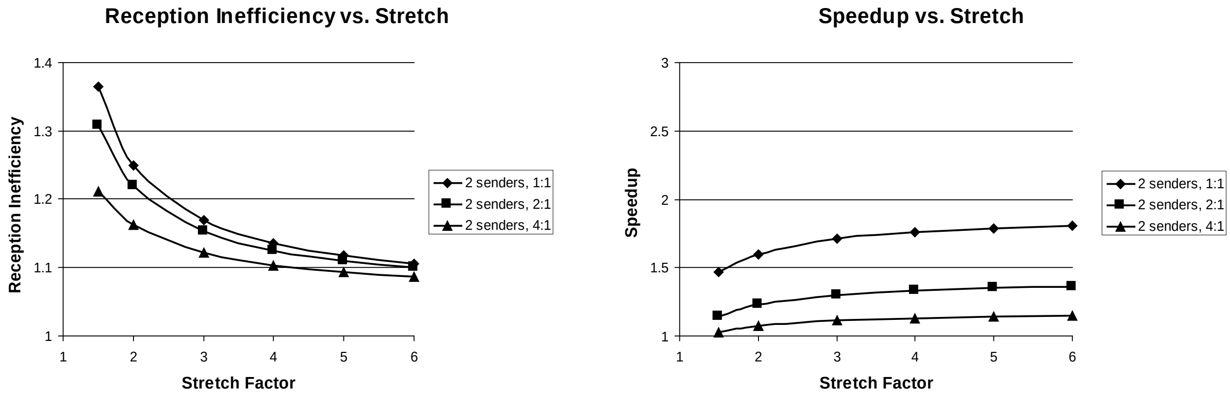

Fig. 2. Reception Efficiency and Speedup vs. Stretch Factor, packets arrive at different rates from two senders, average of 1000 trials

We then solve for z, finding $z = n\left(1 - \sqrt{1 - \frac{m}{n}}\right)$. The expected total number of packets sent is then $2z$, where

$$\begin{aligned} \text{expected packets sent} &= 2z = 2n\left(1 - \sqrt{1 - \frac{m}{n}}\right) \\ &\approx m\left(1 + \frac{m}{4n} + \frac{m^2}{8n^2} + \dots\right), \end{aligned}$$

where the second equation follows from the Taylor series expansion of $\sqrt{1-x}$. Note that $\frac{m}{n} = \frac{\epsilon}{c}$, where ϵ is the decoding inefficiency and c is the stretch factor, so the (approximate) reception inefficiency η can be expressed in terms of ϵ and c:

$$\eta \approx \epsilon\left(1 + \frac{\epsilon}{4c} + \frac{\epsilon^2}{8c^2} + \dots\right).$$

Similarly, if there are three senders of equal rate, then we wish z_3 to be m. Again, using equation (1) as an equality, we find

$$\begin{aligned} m = z + z\left(1 - \frac{z}{n}\right) + z\left(1 - \frac{z + z\left(1 - \frac{z}{n}\right)}{n}\right), \text{ or} \\ \frac{z^3}{n^2} - 3\frac{z^2}{n} + 3z - m = 0. \end{aligned}$$

Some simple algebra yields that in this case $z = n\left(1 - \sqrt[3]{1 - \frac{m}{n}}\right)$.

More generally, it is easy to prove by induction that if there are i senders sending at the same rate, then by using equation (1) as an equality, we find that on average each sender will send $z = n\left(1 - \sqrt[i]{1 - \frac{m}{n}}\right)$ packets. Hence, when the sending rates are equal for i senders, a good approximation for the reception inefficiency is

$$\eta \approx ic\left(1 - \sqrt[i]{1 - \frac{\epsilon}{c}}\right). \quad (2)$$

As an example, with 4 senders and a stretch factor of 2, this formula predicts $\eta = 1.368$, which closely matches our simulation results in the next section.

A.2 Simulation results

To gauge the performance of the no feedback protocol, we examine the results from simulations in Figures 1 and 2. These plots quantify the cost (reception inefficiency) and benefit (speedup) of using the no feedback protocol while varying the stretch factor, the number of senders, and the relative transmission rates. Each data point represents the average of 1000 simulated trials. In each trial, we assume a file comprised of 1000 packets can be decoded once 1055 distinct packets arrive at the receiver. That is, for simplicity we assume a flat 5.5% overhead associated with the Tornado code, and ignore the small variation in the decoding inefficiency. (In Figure 1, this is depicted by the 1 sender line in the plot.) We justify this simplification further shortly. Note that these speedup results include only the time to obtain the packets, and not the decoding time to reconstruct the file, which is minimal.

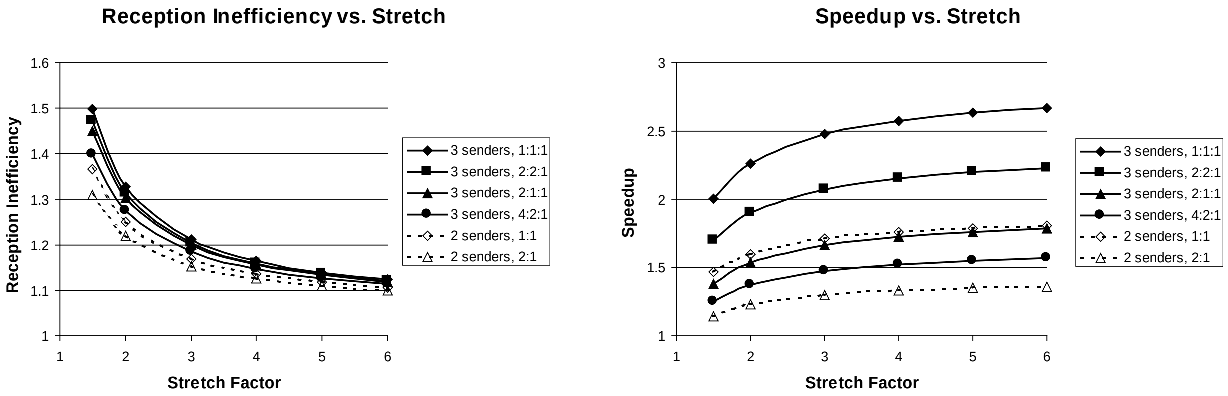

Fig. 3. Reception Efficiency and Speedup vs. Stretch Factor, packets arrive at different rates from three senders, average of 1000 trials

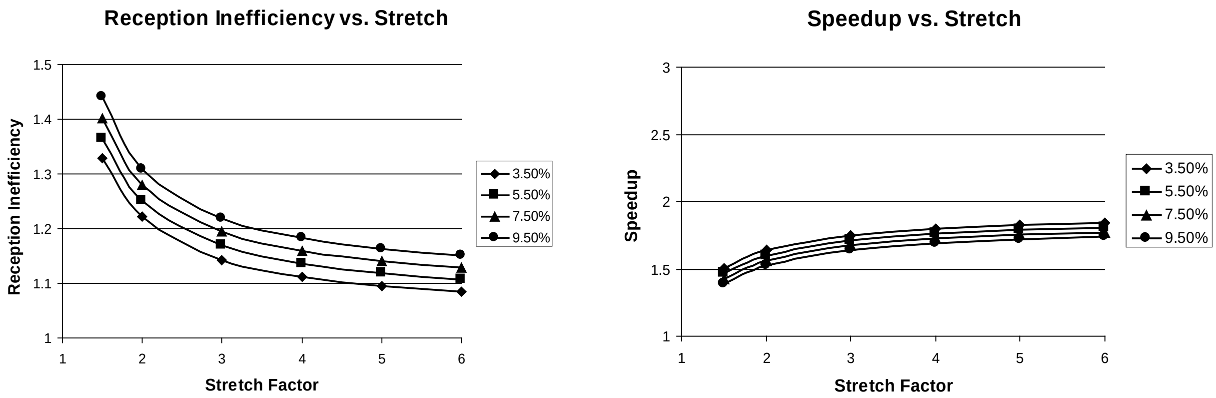

Fig. 4. Reception Efficiency and Speedup vs. Decoding Inefficiency, packets arrive at an equal rate from two senders, average of 1000 trials

In Figures 1 and 2, we vary the stretch factor and the number of senders, assuming packets arrive from each sender at an equal rate. Recall that equal rate transmission maximizes the number of duplicate packets received, and is thus a worst-case assumption for reception inefficiency. The main point these two plots depict is that beyond a moderate stretch factor of $c = 3$, adding additional sources dramatically speeds up downloads with minimal additional reception inefficiency. At $c = 4$, using two senders instead of one almost halves download time, with an overhead of approximately 14%. Using four senders instead of two almost halves download time again with an additional loss in efficiency of less than 4%.

These speedup results hold also in the case where there are losses, in the following sense. Suppose that the packet loss rate along every path is fixed and steady, so that for example one out of every ten packets is dropped. Then the actual arrival rate from each processor remains the same, and as long as the stretch factor is high enough so that a sender need not cycle through its packets multiple times, the same speedup results will hold. Moreover, the presence of burst losses over small timescales also does not substantially alter the speedup results. In the case where a sender does cycle, a receiver may obtain the same packet multiple times from the same sender; we do not consider this here, as we would expect this scenario to be a rare case using a reasonable stretch factor. It is also worth noting that using equation (2) to calculate the reception inefficiency matches the results of our simulations to the nearest thousandth.

In Figure 2, we consider the common case where packets arrive at different rates from two senders. This may happen for a variety of reasons. One server may be more heavily loaded, or one of the paths from server to receiver may be more heavily congested, triggering congestion control mechanisms and lowering the packet transmission rate along that route. Alternatively, the receiver may be using two different media, such as simultaneous use of a modem and an ISDN line. The label 4:1 in Figure 2 means packets from one sender arrive at four times the rate of the other sender. One effect is that the receiver obtains fewer duplicate packets, so the reception inefficiency is reduced by virtue of a reduction in the distinctness inefficiency. Of course, in cases where a second sender provides packets at a much lower rate, the speedup is much less significant. In Figure 2, the speedup given is that over a receiver with a single connection *to the faster sender*. Even in a case where one sender sends packets at a much higher rate, we obtain reasonable speedups accessing multiple senders in parallel.

Figure 3 provides similar simulation results for the case of three senders with differing sending rates. The label 2:2:1, for example, means that there are two fast senders that send at twice the rate of a slower sender. Again, the speedup given is in comparison to a receiver using a single access protocol to attach to the fastest of the senders.

These plots also depict the marginal cost and marginal benefit of a receiver accessing a third sender in addition to the two which it is currently accessing. For example, one would com-

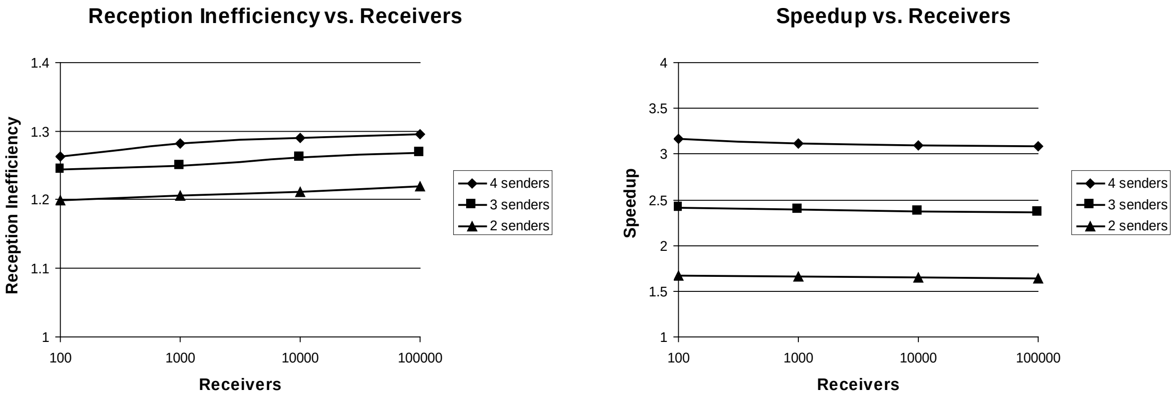

Fig. 5. Worst Case Reception Efficiency and Speedup vs. Receivers, packets arrive at an equal rate from each sender (average of the worst case for 3 trials)

pare the speedup of three senders in a 2:2:1 sending pattern to the speedup of two senders in a 1:1 sending pattern to quantify the benefit of adding a third sender with sending rate half that of the other two. As one would expect, the marginal gain from utilizing a third, slower sender is much less significant than the marginal gain of using two senders instead of one. Still, even in situations where sending rates are disparate, additional senders can still impact downloading performance significantly.

We now justify that when studying quantities such as the average reception inefficiency and average speedup, using the average decoding inefficiency of 5.5% provides an acceptable approximation. Figure 4 shows how the average reception inefficiency and average speedup vary with the decoding inefficiency in the case of two senders sending at equal rates. The variation is relatively minor; this alone somewhat justifies the approximation. Moreover, the reception inefficiency and speedup grow roughly linearly in the decoding inefficiency over this range. This implies that the average reception inefficiency as the decoding inefficiency varies is approximately equal to the reception inefficiency at the average decoding inefficiency, and similarly for the average speedup. Hence using the average decoding inefficiency in our simulations provides good approximations for the actual behavior of the average reception inefficiency and average speedup. The situation is similar for three or four senders as well.

B. Many-to-many distribution

Because of the lack of control information required, the no feedback parallel access approach provides a simple and elegant solution to the *many-to-many* distribution problem. For example, suppose that hundreds or thousands of Internet users all wish to download a new file, and multiple servers are available to send encoded versions of it. Instead of using point-to-point connections, all the senders can cycle through and multicast their encoded version of the file. Each receiver may listen to any number of those transmissions, terminating membership in the sessions when it has obtained enough packets to decode.[3] Because the protocol is stateless and since the receivers can gather packets independently over time and over senders, the protocol scales easily to a vast number of receivers. The use of scalable multicast ensures that the bandwidth requirements of the protocol would be significantly lower than approaches built on point-to-point connections, even if the reception inefficiency of the scheme is moderate. Moreover, congestion control can be implemented with this approach using ideas from layered multicast of [26] and [6].

We demonstrate the scalability and utility of this protocol by measuring worst-case efficiency and speedups in simulations of between 100 and 100,000 users. In Figure 5, we give the worst reception inefficiency and speedup seen for various numbers of receivers, averaged over three trials, using a stretch factor of $c = 3$. That is, for example, we took the average of three different trials of 100,000 users; for each trial, we use the lowest speedup or highest reception inefficiency obtained by any receiver. It is evident that this method scales well to large numbers of users. Indeed, the worst case values are not markedly different from the average values presented in Figure 1. In practice, there would be somewhat more variation in speedup and reception inefficiency due to the fact that the decoding inefficiency is not fixed at 5.5% as we had assumed.

C. Extensions: Solutions with minimal control information

One limitation of the stateless transmission protocols we have described so far is that the lack of coordination between the sources can result in duplicate transmissions and hence distinctness inefficiency. In this section, we describe additional techniques which can be used in conjunction with the existing protocol to limit the magnitude of distinctness inefficiency. The first of these techniques requires a negotiation step at connection set-up time; the second makes use of additional renegotiations during transmission. If it is practical to incorporate these negotiation steps into the implementation, improved performance will result, but they are not essential.

The first enhancement requires all sources to use the same encoding and the same random permutation in transmitting the data associated with a given file. Then, at connection establishment, receivers may request that a given source begin transmission of the source data at a particular offset into the permutation. In the event that the number of sources s that will be used is known to the receiver, a natural way to use this enhancement would have the receiver request offsets corresponding to separate $1/s$ portions of the encoding. The obvious upshot of this

[3] In fact, a network-friendly termination strategy would be to disconnect from all but the highest bandwidth transmission slightly before completion.

approach is that when sources transmit at roughly equal rates, the data which they collectively transmit is guaranteed to contain no duplicates for a significant period of time.

But when sending rates vary, distinctness may drop significantly once one source "laps" another. This drawback can be remedied in part by having a receiver request both a start and end offset at connection set-up time. With this modification, once the source transmits all packets in that range, it is then free to transmit additional packets in a random order of its own choosing. While this extension could significantly improve performance for individual receivers, it appears unsuitable for many-to-many distribution.

A second enhancement uses explicit renegotiations in addition to the initial establishment of offsets. After transmitting all packets from a specified initial range, a source would transmit packets next from a new range sent in a renegotiation message from the receiver, rather than transmitting packets at random. This renegotiation process can be much simpler and more flexible than renegotiation in algorithms which do not use encoding at all, because the receiver does not need to obtain specific packets, but just a sufficiently large set of distinct packets for decoding. A substantial advantage of this approach is that recovery from moderate packet loss is handled explicitly by the protocol, since the encoding itself enables us to recover the file in the event of loss.

Even for a larger number of senders, the rates have to be widely disparate for renegotiations to be necessary. The point is that the redundancy in the encoding immediately gives each of s senders the ability to send a separate c/s fraction of the encoded file; whereas without encoding, each sender is initially responsible for just a $1/s$ fraction of the file, leading to renegotiations under disparate arrival rates. Here, if $c \geq \epsilon s$, where ϵ is the decoding inefficiency, then any one sender can supply enough packets for the receiver to reconstruct the message, in the event that there is no packet loss. In this case, the other senders can be arbitrarily slow compared to one fast sender. Of course, this is only possible for reasonable values of s, since excessively large values of c will introduce large memory requirements and slow down the decoding process. In practice, we expect small values of s to be the norm, so this may well be the common case.

VI. Conclusion

We have considered parallel access to multiple mirror sites and related many-to-many distribution problems. Our approach introduced protocols for these problems using erasure codes, based on the idea of a digital fountain. We suggest that erasure codes are essential for building efficient, scalable protocols for these problems and that Tornado codes in particular provide the necessary encoding and decoding performance to make these protocols viable in practice. The variety of protocols and tunable parameters we describe demonstrates that one can develop a spectrum of solutions that trade off the need for feedback, the bandwidth requirements, the download time, the memory requirements, and the engineering complexity of the solution in various ways. We therefore believe that our straightforward approach will be effective in a variety of settings, including on the Internet and for mobile wireless receivers.

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] K. C. Almeroth, M. Ammar, and Z. Fei, "Scalable Delivery of Web Pages Using Cyclic Best-Effort (UDP) Multicast," In *Proc. of IEEE INFOCOM '98*, San Francisco, 1998.

[3] A. Bestavros. "An adaptive information dispersal algorithm for time-critical reliable communications." In I. Frisch, M. Malek, and S Panwar, editors, *Network Management and Control, Volume II.* Plenum Publishing Corporation, New York, New York, 1994.

[4] A. Bestavros. "AIDA-based real-time fault-tolerant broadcast disks." In *Procedings of the 16th IEEE Real-Time System Symposium,* 1996.

[5] 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.

[6] 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, 1998.

[7] R.L. Carter and M.E. Crovella. "Dynamic Server Selection Using Bandwidth Probing in Wide-Area Networks," In *Proc. of IEEE INFOCOM '97*

[8] 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.

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

[10] C. Hänle. "A Comparison of Architecture and Performance between Reliable Multicast Protocols over the MBONE." Diploma Thesis, Institute of Telematics, University of Karlsruhe. Available at `www.starburstcom.com.w_papers.htm`.

[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] 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.

[13] 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.

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

[15] N. F. Maxemchuk, "Dispersity Routing." In *Proc. of the International Conference on Communications*, pp. 41-10 – 41-13, 1975.

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

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

[18] M. Mitzenmacher, *The Power of two Choices in Randomized Load Balancing*, Ph. D. thesis, University of California at Berkeley, 1996.

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

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

[21] 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.

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

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

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

[25] S. Seshan, M. Stemm, and R. Katz, "SPAND: Shared Passive Network Performance Discovery," In *Proc. of Usenix Symposium on Internet Technologies and Systems '97*, Monterey, CA, December 1997.

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

[27] 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.

[28] 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.