Informed Content Delivery Across Adaptive Overlay Networks

John W. Byers†, Jeffrey Considine†, Michael Mitzenmacher‡, Stanislav Rost§

***Abstract*—Overlay networks have emerged as a powerful and highly flexible method for delivering content. We study how to optimize throughput of large transfers across richly connected, adaptive overlay networks, focusing on the potential of collaborative transfers between peers to supplement ongoing downloads. First, we make the case for an erasure-resilient encoding of the content. Using the digital fountain encoding approach, end-hosts can efficiently reconstruct the original content of size n from a subset of *any* n symbols drawn from a large universe of encoding symbols. Such an approach affords reliability and a substantial degree of application-level flexibility, as it seamlessly accommodates connection migration and parallel transfers while providing resilience to packet loss. However, since the sets of encoding symbols acquired by peers during downloads may overlap substantially, care must be taken to enable them to collaborate effectively. Our main contribution is a collection of useful algorithmic tools for efficient summarization and approximate reconciliation of sets of symbols between pairs of collaborating peers, all of which keep message complexity and computation to a minimum. Through simulations and experiments on a prototype implementation, we demonstrate the performance benefits of our informed content delivery mechanisms and how they complement existing overlay network architectures.**

***Index Terms*—Overlay, peer-to-peer, content delivery, digital fountain, erasure code, min-wise permutation, Bloom filter, reconciliation.**

I. INTRODUCTION

Consider the problem of distributing a large new file across a content delivery network of several thousand geographically distributed machines. Transferring the file with individual point-to-point connections from a single source incurs two performance limitations: wasted bandwidth and transfer rates limited by the characteristics of the end-to-end paths. The problem of excessive bandwidth consumption can be solved by reliable multicast. For example, one elegant and scalable solution is the digital fountain approach [9], whereby the content is first encoded via an erasure-resilient encoding [28], [19], then transmitted to clients. In addition to providing resilience to packet loss, this approach also accommodates asynchronous client arrivals and, if layered multicast is also employed, heterogeneous client transfer rates.

Unfortunately, IP multicast suffers from limited deployment, which has led to the development of *end-system* approaches [11], [14], along with a wide variety of related schemes relevant to peer-to-peer content delivery architectures [31], [10]. These architectures overcome the deployment hurdle faced by IP multicast by constructing *overlay* topologies that comprise collections of unicast connections between end-systems, in which each edge (or connection) in the overlay is mapped onto a path in the underlying physical network by IP routing.

End-system multicast differs from IP multicast in a number of fundamental aspects. First, overlay-based approaches do not use a multicast tree; indeed, they may map multiple virtual connections onto the same network links. Second, unlike IP multicast trees, overlay topologies may flexibly adapt to changing network conditions. For example, applications using overlay networks may reroute around congested or unstable areas of the Internet [1], [30]. Finally, end-systems are now explicitly required to collaborate. This last point is crucial and forms the essence of the motivation for our work: given that end-systems are required to collaborate in overlays, does it necessarily follow that they should operate like routers, and simply forward packets? We argue that this is not the case, and that end-systems in overlays have the opportunity to improve performance provided they are able to actively collaborate, in an informed manner.

We now return to the second limitation mentioned above: the transfer rate to a client in a tree topology is limited by the available bandwidth of the bottleneck link along the path from the server. In contrast, overlay networks can overcome this limitation. In systems with ample bandwidth, transfer rates across overlay networks can substantially benefit from additional cross-connections between end-systems, if the end-systems collaborate appropriately. Assuming that a given pair of end-systems has not received *exactly the same* content, this extra bandwidth can be used to fill in, or *reconcile*, the differences in received content, thus reducing the total transfer time.

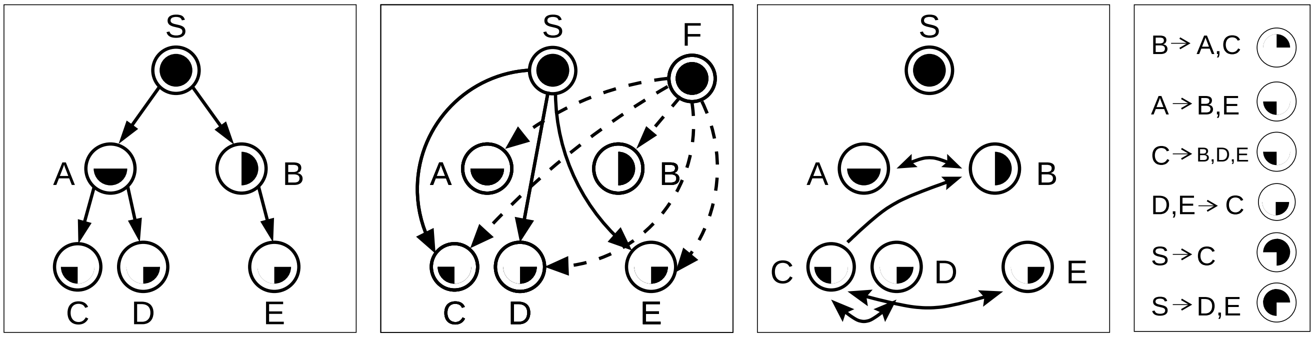
Our approach to addressing these limitations is illustrated in the content delivery scenario of Figure 1. In the initial scenario depicted in Figure 1(a), S is the source and all other nodes in the tree (nodes A through E) represent end-systems downloading a large file via end-system multicast. Each node has a *working set* of packets, the subset of packets it has received (for simplicity, we assume the content is not encoded in this example). Even if the overlay management of the end-system multicast architecture ensured the best possible embedding of the virtual graph onto the network graph (for some appropriate definition of best), there is still considerable room for improvement. A first improvement can be obtained by harnessing the power of *parallel downloads* [8], i.e. establishing concurrent connections to multiple servers or peers with complete copies of the file (Figure 1(b)). More generally, additional significant performance benefits may be obtained by taking advantage of "perpendicular" connections between nodes whose working sets are complementary, as pictured in Figure 1(c). Benefits of establishing concurrent connections to multiple peers have been demonstrated by popular peer-to-peer file sharing systems such as Kazaa, Grokster and Morpheus. The improvements in transfer rates that these programs obtain provide preliminary evidence of availability of bandwidth for opportunistic downloads between collaborating peers.

As discussed earlier, the tree and directed acyclic graph topologies of Figures 1(a) and 1(b) impede the full flow of con-

†Boston University, {byers, jconsidi}@cs.bu.edu. Supported in part by NSF grants ANI-0093296 and ANI-9986397.

‡Harvard University, michaelm@eecs.harvard.edu, supported in part by NSF grants CCR-9983832, CCR-0118701, CCR-0121154, and an Alfred P. Sloan Research Fellowship.

§MIT, stanrost@csail.mit.edu



(a) Initial delivery tree. (b) Parallel downloads. (c) Collaborative transfers. (d)

Fig. 1. **Possibilities for content delivery.** Shaded content within a node in the topology represents the working set of that node. Connections in (b) supplement those in (a); connections in (c) supplement those in (a)+(b). Source S and peer F have the content in its entirety. A, B each have different, but overlapping halves of the full content. C, D, E each have 25% of the content. Panel (d) depicts the portions of content which can be beneficially exchanged via pairwise transfers.

tent to downstream receivers, as the rate of flow monotonically decreases along each end-system hop on paths away from the source. In contrast, the opportunistic connections of the graph of Figure 1(c) allow for higher transfer rates, but simultaneously demand more careful orchestration between end-systems to achieve those rates. In particular, any pair of end-systems in a peer-to-peer relationship must be able to determine and transfer those packets that lie in the set difference of their working sets.

When working sets are limited to small groups of contiguous blocks of sequentially indexed packets, reconciliation is simple, since each block can be succinctly specified by an index and a size. However, restricting working sets to such simple patterns greatly limits flexibility to the frequent changes which arise in adaptive overlay networks, as we argue in Section II. But the added flexibility provided by encoded content also makes reconciliation a more difficult problem. To address this challenge, in Sections III-VI, we provide a set of tools for estimating, summarizing, and approximately reconciling working sets of connected clients, all of which keep message complexity and computation to a minimum. In Section VII, we demonstrate through simulations and experiments on a prototype implementation that these tools, coupled with the encoding approach, form a highly effective delivery method which can substantially reduce transfer times over existing methods. We outline ongoing research directions and draw conclusions in Sections VIII and IX.

II. Content Delivery Across Overlay Networks

We motivate our approach by sketching fundamental challenges that must be addressed by any content delivery architecture and outlining the set of opportunities that an overlay approach affords. Then, we argue the pros and cons of encoded content, namely a small amount of added complexity for greatly improved flexibility and scalability.

A. Challenges and Opportunities

The fluid Internet environment poses a number of challenges that a content delivery infrastructure must cope with, including:

- **Asynchrony:** Receivers may open and close connections or leave and rejoin the infrastructure at arbitrary times.
- **Heterogeneity:** Connections vary in speed and loss rates.
- **Transience:** Routers, links and end-systems may fail and their performance may fluctuate over time.
- **Scalability:** The service must scale to large receiver populations and large content.

For example, a robust overlay network should have the ability to adaptively detect and avoid congested or temporarily unstable areas of the network [1], [30]. It should also be able to dynamically establish paths with the most desirable end-to-end characteristics. Such reactive behavior of the virtual topology may frequently induce the nodes to reconnect to better-suited peers. However, this adaptive behavior exacerbates the problems enumerated above. Another consequence of the fluidity of the environment is that content is likely to be disseminated non-uniformly across peers. For example, discrepancies between working sets may arise due to uncorrelated losses, bandwidth differences, asynchronous joins, and topology reconfigurations. More specifically, receivers with higher transfer rates and receivers who initiate the download earlier will simply have more content than their peers.

Finally, we also want to take advantage of a significant *opportunity* presented by overlay networks discussed in the introduction: the ability to download content across multiple connections in parallel. More generally, we wish to make beneficial use of any available connection present in an adaptive overlay, including ephemeral connections which may be short-lived, may be preempted, or whose performance may fluctuate over time. This opportunity raises the further challenge of delivering content which is not only useful, but which is useful even when other connections are being employed in parallel, and doing so with a minimum of set-up overhead and message complexity.

B. Limitations of Stateful Solutions

While issues of connection migration, heterogeneity, and asynchrony are tractable, solutions to each problem generally require significant per-connection state. The retained state makes such approaches highly unscalable. Moreover, bulky per-connection state can have significant impact on performance, since this state must be maintained in the face of reconfiguration and reconnection.

Parallel downloading using stateful approaches is by itself problematic, as discussed in [8]. The natural approach divides the missing packets into disjoint sets that can be downloaded from different sources. But network heterogeneity and transience necessitate frequent renegotiation of which packets to obtain from each source. Also, there is a natural bottleneck that arises from the need to obtain "the last few packets" [8]. Both of these problems are alleviated by the use of encoded content, as we describe below. While we do not argue that parallel downloading with unencoded content is impossible (for example, see [29]), the use of encoding facilitates simpler and more effective parallel downloading.

One other complication is that in our framework, it is actually *beneficial* to have partially downloaded content distributed unevenly across participating end-systems, so that there is considerable discrepancy between working sets. As noted earlier, discrepancies in working sets will arise naturally. Stateful approaches in which end-systems attempt to download contiguous blocks of unencoded packets work against this goal, since end-systems effectively strive to reduce the discrepancies between the packets they obtain. Again, in schemes using encoded content, this problem is not a consideration.

C. Benefits of Encoded Content

An alternative to using stateful solutions is the use of the digital fountain paradigm [9] running over an unreliable transport protocol. The digital fountain approach was originally designed for point-to-multipoint transmission of large files over lossy channels. Resilience to packet loss is achieved by using an *erasure correcting code* [19], [28] to produce an unbounded stream of encoding symbols derived from the source file. The encoding stream has the guarantee that a receiver is certain to be able to recover the original source file from *any* subset of distinct symbols in the encoding stream equal to the size of the original file. In practice, this strong decoding guarantee is relaxed in order to provide efficient encoding and decoding times. Current implementations are capable of efficiently reconstructing the file having received only a few percent more than the number of symbols in the original file [19], [18], [9]. A digital fountain approach provides a number of important benefits: which are useful in a variety of content delivery scenarios:

- **Continuous Encoding:** Senders with a complete copy of a file may continuously produce fresh encoding symbols.
- **Time-Invariance:** New encoding symbols are produced independently from symbols produced in the past.
- **Tolerance:** Digital fountain streams are useful to all receivers regardless of the times of their connections or disconnections and their rates of sampling the stream.
- **Additivity:** Fountain flows generated by senders with different sources of randomness are uncorrelated, so parallel downloads from multiple servers with complete copies of the content require no orchestration.

While the full benefits of encoded content described above apply primarily to a source with a copy of the entire file, some benefits can be achieved by end-systems with partial content, by re-encoding the content as described in Section V. The flexibility provided by the use of encoding frees the receiver from receiving all of a set of distinct symbols and enables fully stateless connection migrations. It also allows the nodes of the overlay topology to connect to as many senders as necessary and obtain distinct encoding symbols from each, provided these senders are in possession of the entire file.

There is one significant disadvantage from using encoded content, aside from the small overhead associated with encoding and decoding operations. In a scenario where encoding symbols are drawn from a large universe, end-systems that hold only part of the content must take care to arrange transmission of useful information between one another. The digital fountain approach handles this problem in the case where an end-system has decoded the entire content of the file; once this happens, the end-system can generate fresh encoded content at will. However, when collaborating end-systems have only a portion of the content, reconciliation methods are needed to avoid redundant transmissions.

III. Reconciliation and Informed Delivery

The preceding sections have established expectations for informed collaboration. We abstract our solutions away from the issues of optimizing the layout of the overlay over time [11], [14], [1], as well as distributed naming and indexing; our system supplements any solutions employed to address these issues.

The approaches to reconciliation we propose are local in scope, and typically involve a pair or a small number of end-systems. In the setting of wide-area content delivery, many pairs of systems may desire to transfer content in an informed manner. For simplicity, we will consider each such pair independently, although we point to the potential use of our techniques to perform more complex, non-local orchestration. Our goal is to provide the most cost-effective reconciliation mechanisms, measuring cost both in computation and message complexity. In the subsequent sections, we propose the following approaches:

Coarse-grained reconciliation employs *sketches* of each peer's working set. Using random sampling or min-wise permutations [5], coarse-grained reconciliation is not resource-intensive and allows quick estimates of the fraction of symbols common to the working sets of both peers.

Speculative transfers involve a sender performing "educated guesses" as to which symbols to generate and transfer. As the number of symbols common to both working sets increases, there are fewer useful symbols which the sender can generate cheaply, and the sender must strike a balance between increased computation and the probability of utility. This process can be fine-tuned using the results of coarse-grained reconciliation.

Fine-grained reconciliation employs compact, searchable working set summaries such as Bloom filters [3] or approximate reconciliation trees (to be introduced in Section VI). Fine-grained approaches are more resource-intensive and allow a peer to determine the symbols in the working set of another peer with a quantifiable degree of certainty.

The techniques we describe provide a range of options and are useful in different scenarios, primarily depending on the resources available at the end-systems, the correlation between the working sets at the end-systems, and the requirements of precision. The sketches can be thought of as an end-system's calling card: they provide some useful high-level information, are extremely lightweight, can be computed efficiently, can be incrementally updated at an end-system, and fit into a single 1KB packet. Generating the searchable summaries requires a bit more effort: while they can still be computed efficiently and incrementally updated, they require more space, and a gigabyte of content will typically require a summary on the order of 1MB in size. Finally, speculative transfers tune the content sent across a peer-to-peer connection based on information conveyed in sketches.

IV. Coarse-grained Reconciliation

We start with simple and quick methods for coarse-grained reconciliation which estimate the resemblance of the working sets of pairs of nodes prior to establishing connections. Knowledge of the resemblance allows a receiver to determine the extent to which a prospective peer can offer useful content. Our methods are designed to give accurate answers when only a single 1KB packet of data is transferred between peers.

We first establish the framework and notation. Let peers A and B have working sets S_A and S_B containing symbols from an encoding of the file.

Definition 1 (Containment and Resemblance) The *containment* of B in A is the quantity $\frac{|S_A \cap S_B|}{|S_B|}$. The *resemblance* of A and B is the quantity $\frac{|S_A \cap S_B|}{|S_A \cup S_B|}$.

These definitions are due to Broder [4] and were applied to determine the similarity of documents in search engines. The containment represents the fraction of elements B that are useless (already known) to A. If this quantity is close to zero, the containment is small, and B rates to be a useful source of information for A. We point out that containment is not symmetric, while resemblance is. Also, given $|S_A|$ and $|S_B|$, an estimate for one can easily be used to calculate an estimate for the other.

We suppose that each element of a working set is identified by an integer key; sending an element entails sending its key. We will think of these keys as unique, although they may not be; for example, if the elements are determined by a hash function seeded by the key, two keys may generate the same element with small probability. This may introduce small errors in estimating the containment, but since we generally care only about the approximate magnitude of the containment, this will not have a significant impact. With 64-bit keys, a 1KB packet can hold roughly 128 keys, which enables reasonable estimates for the techniques we describe. Finally, we assume that the integer keys are distributed over the key space uniformly at random, since the key space can always be transformed by applying a (pseudo)random hash function.

The first approach we consider is straightforward random sampling: simply select k elements of the working set at random (with replacement) and transport those to the peer. (We may also send the size of the working set, although this is not essential.) Suppose A sends B a random sample K_A from S_A. The probability that each element in K_A is also in S_B is $\frac{|S_A \cap S_B|}{|S_B|}$, and hence $\frac{|K_A \cap S_B|}{k}$ is an unbiased estimate of the containment. Random samples can be incrementally updated upon acquisition of new elements using reservoir sampling [32]. Random sampling suffers the drawback that B must search for each element of K_A in its own list S_B. Although such searches can be implemented quickly using standard data structures (interpolation search will take $O(\log\log|S_B|)$ average time per element), they require some extra updating overhead. One remedy, suggested in [4], is to sample only those elements whose keys are 0 modulo k for an appropriately chosen k, yielding samples K_A and K_B. (Here we specifically assume that the keys are random.) In this case, $\frac{|K_A \cap K_B|}{|K_B|}$ is an unbiased estimate of the containment; moreover, all computations can be done directly on the small samples, instead of on the full working sets. However, this technique generates samples of variable size, which can be awkward, especially when the size of the working sets varies dramatically across peers. Another concern about both of these random sampling methods is that they do not easily allow one peer to check the resemblance between prospective peers. For example, if peer A is attempting to establish connections with peers B and C, it might be helpful to know the resemblance between the working sets of B and C.

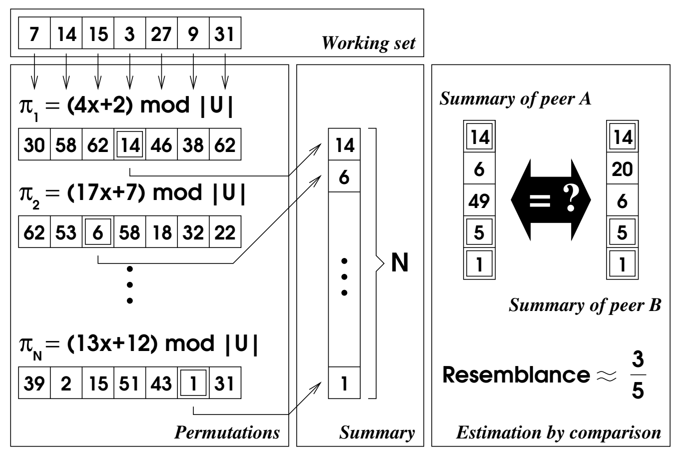


Fig. 2. Example of min-wise summarization and estimation of resemblance (Key universe size is 64, example permutation functions shown).

Another clever sampling technique from [4] avoids the drawbacks of the first two approaches. This approach, which we employ, calculates working set resemblance based on *min-wise sketches*, following [4], [5]; the method is depicted in Figure 2. Let π_j represent a random permutation on the key universe U. For a set $S = \{s_1, s_2, \ldots, s_n\}$, let $\pi_j(S) = \{\pi_j(s_1), \pi_j(s_2), \ldots, \pi_j(s_n)\}$, and let $\min \pi_j(S) = \min_k \pi_j(s_k)$. Then for two working sets S_A and S_B containing symbols of the file F, we have $x = \min \pi_j(S_A) = \min \pi_j(S_B)$ if and only if $\pi_j^{-1}(x) \in S_A \cap S_B$. That is, the minimum element after permuting the two sets S_A and S_B matches only when the inverse of that element lies in both sets. In this case, we also have $x = \min \pi_j(S_A \cup S_B)$. If π_j is a random permutation, then each element in $S_A \cup S_B$ is equally likely to become the minimum element of $\pi_j(S_A \cup S_B)$. Hence we conclude that $\min \pi_j(S_A) = \min \pi_j(S_B)$ with probability $r = \frac{|S_A \cap S_B|}{|S_A \cup S_B|}$. Note that this probability is the resemblance of A and B. To estimate the resemblance, peer A computes $\min \pi_j(S_A)$ for some fixed number of permutations π_j, and similarly for B and S_B. The peers must agree on these permutations in advance; we assume they are fixed universally off-line.

For B to estimate $\frac{|S_A \cap S_B|}{|S_A \cup S_B|}$, A sends B a vector containing A's minima, $v(A)$. B then compares $v(A)$ to $v(B)$, counts the number of positions where the two are equal, and divides by the total number of permutations, as depicted in Figure 2. The result is an unbiased estimate of the resemblance r since each position is equal with probability r.

In practice, truly random permutations cannot be used, as the storage requirements are impractical. Instead, we may use simple permutations, such as $\pi_j(x) = ax + b \pmod{|U|}$ for randomly chosen a and b and when U is prime, without affecting overall performance significantly [5]. Also, instead of sending values in $\pi_j(x) \in U$, these values can be hashed to fewer bits, allowing us to store more sketch elements in each packet. This introduces the possibility of false positives, which can subsequently be accounted for when estimating the resemblance. We omit details for lack of space.

The min-wise sketches above allow similarity comparisons given any two sketches for any two peers. Moreover, these sketches can be combined in natural ways. For example, the sketch for the union of S_A and S_B is easily found by taking the coordinate-wise minimum of $v(A)$ and $v(B)$. Estimating the resemblance of a third peer's working set S_C with the combined working set $S_A \cup S_B$ can therefore be done with $v(A), v(B)$, and $v(C)$. Min-wise sketches can also be incrementally updated upon acquisition of new content, with constant overhead per receipt of each new element.

V. Speculative Reconciliation

Recall that speculative transfers involve a sender performing "educated guesses" as to which symbols to generate and transfer. For example, after exchanging min-wise summaries, peers may know that there is an abundance of useful symbols to transfer, but they do not yet know *which ones* are useful. In this case, one peer wishes to send symbols which are *probably* useful to the other, while attempting to minimize computation and communication costs. In some cases, such as when the containment of B in A is low, speculative reconciliation is trivial, since most of B's symbols are useful to A. But when the containment of B in A is high, this simple strategy is inefficient. For example, consider peers A and B that estimate the containment of B in A to be 0.8. Without fine-grained reconciliation, there is an 80% probability that any symbol that B sends to A is already known and thus is useless. On the other hand, if B randomly chooses 9 encoding symbols and sends the bitwise XOR of them, there is only a probability of $0.8^9 \approx 14\%$ that all 9 encoding symbols were already known to A. This is the intuition behind recoding.

A. Sparse Parity Check Codes

A key ingredient we use in our recoding methods are sparse parity check codes. The content being sent by the encoder is a sequence of symbols $\{x_1, \ldots, x_\ell\}$, where each x_i is called an *input* symbol. An encoder produces a sequence of *encoding symbols* $y_1, y_2, \ldots$ from the set of input symbols. With parity-check codes, each encoding symbol is simply the bitwise XOR of a specific subset of the input symbols. A decoder attempts to recover the original content from the encoding symbols. For a given symbol, we refer to the number of input symbols used to produce the symbol as its *degree*, i.e. $y_3 = x_3 \oplus x_4$ has degree 2. Using the methods described in [19], the time to produce an encoding symbol is proportional to the degree of the encoding symbol, while decoding takes time proportional to the total degree of the symbols in the sequence. Encoding and decoding times are a function of the *average* degree; when the average degree is constant, we say the code is sparse. Well-designed sparse parity check codes typically require recovery of a few percent (less than 5%) of symbols beyond ℓ, the minimum needed for decoding. The *decoding overhead* of a code is defined to be ϵ_d if $(1 + \epsilon_d)\ell$ encoding symbols are needed on average to recover the original content. (There is also a small amount of overhead for the space needed in each packet to identify which input symbols were combined, which is typically represented by a 64-bit random seed.)

Provably good degree distributions for sparse parity check codes were first developed and analyzed in [19]. However, these codes are fixed-rate, meaning that only a pre-determined number of encoding symbols are generated, typically only $c\ell$, where $c > 1$ is a small constant. In some applications such as in [8], this can lead to inefficiencies as servers are forced to retransmit symbols. Similarly, in our application, fixed rates result in reduced diversity in the working sets of peers. Newer codes, called *rateless* codes, avoid this pitfall and allow unbounded numbers of encoding symbols to be generated on demand. Two examples of rateless codes, along with further discussion of the merits of ratelessness, may be found in [18], [21]. Both of these codes also have have strong probabilistic decoding guarantees, along with low decoding overheads and average degrees.

B. Recoding Methods

We now describe recoding for speculative reconciliation. In this setting peers A and B have working sets of encoding symbols. Peer B may attempt to send useful information to peer A by using recoding symbols. A recoding symbol is simply the bitwise XOR of a set of encoding symbols. Like a regular encoding symbol, a recoding symbol must be accompanied by a specification of the encoding symbols blended to create it. But unlike a regular encoding symbol, a recoding symbol must explicitly list the random seeds of the encoding symbols from which it was produced. This has an important side effect of imposing a fixed degree limit upon recoding symbols. Encoding and decoding of recoding symbols are performed in a fashion analogous to the substitution rule. For example, a peer with encoding symbols y_5, y_8 and y_{13} can generate recoding symbols $z_1 = y_{13}$, $z_2 = y_5 \oplus y_8$ and $z_3 = y_5 \oplus y_{13}$. A peer that receives z_1, z_2 and z_3 can immediately recover y_{13}. Then by substituting y_{13} into z_3, the peer can recover y_5, and similarly can recover y_8 from z_2. As the encoding symbols are recovered, the normal decoding process proceeds.

As with normal sparse parity check codes, irregular degree distributions work well for recoding. To generate a good degree distribution for recoding, we modified the degree distribution generation process of [12]. Here, the key modifications are 1) to impose an upper bound on the symbol degree and 2) to generate an encoding for the case in which a set of input symbols have already been recovered by the decoder and where the decoder need not recover all input symbols.

To provide brief intuition about the methods of [12], we consider the probability that a recoding symbol is immediately useful. Suppose peer B knows the exact value of the containment $c = \frac{|S_A \cap S_B|}{|S_B|}$. The probability that a recoding symbol of degree d immediately yields a new encoding symbol is $\frac{\binom{c|S_B|}{d-1}\binom{(1-c)|S_B|}{1}}{\binom{|S_B|}{d}}$. This is maximized for *target degree* $d = \left\lceil \frac{c}{1-c} \right\rceil$. Note that as recoding symbols are received, containment naturally increases and the target degree increases accordingly. Using this formula for d maximizes the probability of immediate benefit but is actually not optimal, since a recoding symbol of this degree runs a large risk of being a duplicate. A theoretical foundation for the design of ideal degree distributions for sparse parity-check codes can be found in [18].

VI. Fine-grained Reconciliation

As shown in Section IV, a single packet can allow peers to estimate the resemblance in their working sets. If the difference is sufficiently large to allow useful exchange of data, the peers may then use the methods of Section V to determine what to exchange. As this level of summarization may be too coarse to be effective, we now describe methods for fine-grained reconciliation that still have an overhead of only a handful of packets.

Fine-grained reconciliation is a set difference problem. Again, suppose peer A has a working set S_A and peer B has a working set S_B, both sets being drawn from a universe U with $|U| = u$. Peer A sends peer B some message M with the goal of peer B determining as many elements in the set $S_B - S_A$ as possible. (For convenience later, we will use $d = |S_B - S_A|$, though d will generally not be known exactly.)

The set difference problem has been widely studied in communication complexity. The focus, however, has generally been on determining the *exact* difference $S_B - S_A$. With encoded content, a peer does not generally need to acquire all of the symbols in this difference. For example, two peers may each have $3\ell/4$ symbols, with $\ell/4$ symbols in common, where ℓ symbols are necessary to reconstruct the file. In this case, only $\ell/4$ of the $\ell/2$ symbols in $S_B - S_A$ need to be transferred. One of our contributions is this insight that *approximate* reconciliation of the set differences is sufficient for our application.

We consider the following performance criteria in our discussion of approaches to reconciliation:

- **Number of communication rounds.**
- **Message size:** the number of bits sent by A to B.
- **Construction time:** for A, the construction time is the time for A to compute its messages; for B, the construction time is the time to produce a representation of S_B.
- **Reconciliation time:** the time for B to compute the approximation to $S_B - S_A$ given the message from A and an appropriate representation of S_B.
- **Accuracy:** the probability that a given element in $S_B - S_A$ is correctly identified by B.

Optimization of the evaluation metrics presented above is especially important in the reconciliation of large data sets. In our application, sets of encoding symbols for 1GB files may have sizes on the order of millions, with hundreds of thousands of differences between overlay peers. It is also desirable to separate costs of pre-processing from reconciliation, since the latter is always incurred in real-time. This is important in a peer-to-peer setting, where a peer reconciling with multiple other peers may wish to amortize the cost of pre-processing across several pair-wise communications. Thus, while we account for all costs, we emphasize the real-time costs of reconciliation. We also note that minimizing communication complexity may be less important as long as the cost for reconciliation messages is small relative to the size of the transfer. With these factors in mind, approximate solutions are highly desirable as they will be both faster and more compact.

The best exact reconciliation protocols to date in terms of communication complexity are based upon the use of characteristic polynomials [23], [25], [24]. A technical description is beyond the scope of this paper, but the main idea involves each peer evaluating the characteristic polynomial of their set at several points and using this information to compute the ratio of the two characteristic polynomials and hence their set differences. One requirement for these protocols is an upper bound on the number of set differences, so as to determine how many points to evaluate. Each of these points must then be updated each time a new element is added to the set. In the case that an upper bound on d is not known, a guess $\bar{d}$ is made and a security parameter k is chosen. Then, $\bar{d} + k$ coefficients are sent, and a probabilistic check is made that will discover if $\bar{d} < d$ with probability $1 - \left(\frac{|S_A|+|S_B|}{u}\right)^k$. While very attractive for small numbers of differences, this method is computationally expensive (cubic) for large numbers of differences, both in construction time and reconciliation time. While the variant in [24] is much faster, it still requires $\Theta(\log d)$ rounds of communication, which is not well suited to our application. This leads us to consider approximate methods which offer much lower computational complexities. Since we are focused on reconciling large number of differences, often $\Omega(|S_A|)$ of them, the information theoretic communication complexity of exact reconciliation is $\Omega(|S_A| \log u)$ in the common case where U is much larger than S_A. Using approximate methods, we can reconcile a constant fraction of the set differences using a message of size $O(|S_A|)$. Table I gives a high level comparison of the various approaches we discuss.

A. Enumeration-based Approaches

In contrast to polynomial-based methods, the simplest, but most expensive approach to computing differences is for peer A to enumerate and send the entire working set S_A. This requires $O(|S_A| \log u)$ bits to be transmitted. A natural alternative is to use hashing. Suppose the set elements are hashed using a random hash function into a universe $U' = [0, h)$. Peer A then hashes each element and sends the set of hashes instead of the actual working set S_A. Now only $O(|S_A| \log h)$ bits are transmitted. Strictly speaking, this process may not yield the exact difference: there is some probability that an element $x \in S_B \setminus S_A$ will have the same hash value as an element y of S_A, in which case peer B will mistakenly believe $x \in S_A$. The miss probability can be made inversely polynomial in $|S_A|$ by setting $h = \text{poly}(|S_A|)$, in which case $\Theta(|S_A| \log |S_A|)$ bits are sent. Using hash tables to compare the lists of elements or hash values sent, exact reconciliation (subject to hash collisions) can

Method	Total Message Size	Construction Time for A	Construction Time for B	Rounds	Reconciliation Time (expected)	Exact?
Polynomial [25]	$O(d \log u)$	$O(d\,\lvert S_A\rvert)$	$O(d\,\lvert S_B\rvert)$	1	$O(d^3)$	yes
Enumeration	$O(\lvert S_A\rvert \log u)$	$O(\lvert S_A\rvert)$	$O(\lvert S_B\rvert)$	1	$O(\lvert S_A\rvert + \lvert S_B\rvert)$	yes
Bloom filter (VI.A)	$O(\lvert S_A\rvert)$	$O(\lvert S_A\rvert)$	$O(\lvert S_B\rvert)$	1	$O(\lvert S_B\rvert)$	no
Merkle tree [23]	$O(d \log h \log \lvert S_B\rvert)$	$O(\lvert S_A\rvert \log \lvert S_A\rvert)$	$O(\lvert S_B\rvert \log \lvert S_B\rvert)$	$O(\log \lvert S_B\rvert)$	$O(d \log \lvert S_B\rvert)$	w.h.p.
Poly/tree hybrid [24]	$O(d\,k \log u)$	$O(\lvert S_A\rvert \log \lvert S_A\rvert)$	$O(\lvert S_B\rvert \log \lvert S_B\rvert)$	$O(\log d)$	$O(d)$	w.h.p.
ART (VI.C)	$O(\lvert S_A\rvert)$	$O(\lvert S_A\rvert \log \lvert S_A\rvert)$	$O(\lvert S_B\rvert \log \lvert S_B\rvert)$	1	$O(d \log \lvert S_B\rvert)$	no

TABLE I

COMPARISON OF VARIOUS FINE-GRAINED RECONCILIATION METHODS WITH AT LEAST CONSTANT ACCURACY.

then be done in $O(|S_A| + |S_B|)$ time with high probability.

A more compact enumeration-based method for approximate reconciliation is to use a Bloom filter [3]. This solution is surprisingly effective, particularly when the number of differences is a large fraction of the set size. A Bloom filter is used to represent a set $S = \{s_1, s_2, \ldots, s_n\}$ of n elements from a universe U of size u, and consists of an array of m bits, initially all set to 0. A Bloom filter uses k independent random hash functions $h_1, \ldots, h_k$ with range $\{0, \ldots, m-1\}$. For each element $s \in S$, the bits $h_i(s)$ are set to 1 for $1 \leq i \leq k$. To check if an element x is in S, we check whether all $h_i(x)$ are set to 1. If not, then clearly x is not a member of S. If all $h_i(x)$ are set to 1, we assume that x is in S, although we are wrong with some probability. Hence a Bloom filter may yield a *false positive*, when it reports that an element x is in S even though it is not in fact in S. The probability of a false positive f depends on the number of bits used per item, m/n, and the number of hash functions k as follows: $f \approx (1 - \mathrm{e}^{-km/n})^k$. (We note that this is a highly accurate approximation and we treat it as an equality henceforth.) This false positive rate is minimized by picking $k = (\ln 2)(m/n)$ which results in $f = (1/2)^{(\ln 2)(m/n)}$.

For an approximate reconciliation solution, peer A sends a Bloom filter F_A of S_A; peer B then simply checks for each element of S_B in F_A. When a false positive occurs for an element $y \in S_B$, peer B will incorrectly assume that peer A also has element y, and thus peer B fails to identify y as an element of $S_B - S_A$. However, the Bloom filter never causes peer B to mistakenly find an element to be in $S_B - S_A$ when it is not.

With Bloom filters, the message size can be kept small while still achieving high accuracy. For example, using just four bits per element of S_A, i.e. a message of length $4|S_A|$, and three hash functions yields an accuracy of 85.3%; using eight bits per element and five hash functions yields an accuracy of 97.8%. Further improvements can be had by using the recently introduced compressed Bloom filter, which reduces the number of bits transmitted between peers at the cost of using more bits to store the Bloom filter at the end-systems and requiring compression and decompression at the peers [26]. With a constant number of hash functions and making the standard assumption that hashes and array accesses are constant time operations, A's construction time is $O(|S_A|)$ to set up the Bloom filter, and B's reconciliation time is $O(|S_B|)$ to find the set difference. B has no construction time, although B could precompute hashes of its set elements to shift reconciliation time to construction time.

The requirement for $O(|S_A|)$ construction time and $O(|S_A|)$ message size may seem excessive for large $|S_A|$. There are several possibilities for scaling this approach up to larger set sizes. For example, for large $|S_A|$ or $|S_B|$, peer A can create a Bloom filter only for elements of S_A that are equal to β modulo γ for some appropriate β and γ. Peer B can then only use the filter to determine elements in $S_B - S_A$ equal to β modulo γ (still a relatively large set of elements). The Bloom filter approach can then be pipelined by incrementally providing additional filters for differing values of β as needed. This pipelining approach can similarly be used in many other schemes.

An unavoidable drawback of Bloom filters is the large reconciliation time: even when the set difference $S_B - S_A$ is small, every element of S_B must be tested against the filter F_A. As described in the introduction, minimizing the reconciliation time can be crucial in applications where reconciliation must be performed in real-time. Our approximate reconciliation tree data structure described in Section C avoids this problem.

B. Search-based Approaches

In contrast to enumeration-based approaches, search-based approaches are more efficient both at finding small numbers of differences and at identifying sets of differences. Search-based approaches to set reconciliation generally leverage Patricia tries [16] and Merkle trees [22]. Patricia tries are used to provide structured searching based upon comparable subsets; while Merkle trees solve the problem of testing the equivalence of large subsets in constant time.

Abstractly, a trie is a complete binary tree that represents a subset S of a universe $U = \{0, \ldots, u-1\}$ (stored at the root) and where the jth child at depth k corresponds to the set $S \cap [(j-1) \cdot u/2^k, j \cdot u/2^k - 1]$. By this definition, a given element x in U is present in all of the nested subsets along a path from the root to the leaf representing the singleton set $\{x\}$. Therefore, when sets S_A and S_B are each represented in trie form, an element in the set difference can be located by identifying a root-to-leaf path where each of the nodes on the path in S_A correspond to distinct subsets from those stored in the nodes on the path in S_B. In this basic method, identifying a single discrepancy involves performing $2 \log u$ comparisons to identify a path of length $\log u$ to a leaf.

However, there are two inefficiencies in the approach described so far. First, the trie has $\Theta(u)$ nodes and depth $\Theta(\log u)$, which is unsuitable when the universe is large. However, almost all the nodes in the trie correspond to the same sets. In fact there are only $2|S| - 1$ non-trivial nodes corresponding to distinct subsets. The trie can therefore be *collapsed* by removing edges between nodes that correspond to the same set, leaving only $2|S|-1$ nodes, and the result is a Patricia trie. An example of a collapsed Patricia trie is depicted in panel (a) of Figure 3.

The second inefficiency is that each of the subsets being compared may be arbitrarily large. It is not difficult to construct Patricia tries with $\Omega(|S|)$ subsets of size $\Omega(|S|)$. Merkle trees [22] provide a convenient probabilistic answer to this challenge. Merkle trees provide a method for signing and comparing large databases while allowing fast updates and identifying differences. For our application, we can form a Merkle tree on top of a trie by concisely representing each node's subset as a single value. At the leaves of the Merkle tree, the value is obtained by applying a hash function to the singleton set stored at the leaf of the trie. The values of internal nodes of a Merkle tree are then obtained by applying a hash function to the values of their children. Using this construction, the hash value stored at a node is a function of all of the elements in its subtree. An example of Merkle hashing applied to a Patricia trie is depicted in panel (c) of Figure 3 using the hash function in panel (b). If the Merkle hashing idea is used in conjunction with tries, the search method described before still applies, with the one difference that hash values instead of whole sets are compared. This affords constant time comparisons, but introduces the risk of false positive matches due to hash collisions at any level of the tree.

An interactive algorithm based upon this approach appeared in [23]. During each round of the algorithm, an additional level of the tree was traversed from any nodes detecting differences. Unfortunately, this requires a logarithmic number of rounds even with randomization and balancing. A hybrid protocol combining these ideas with the polynomial based approach was given in [24] and uses fewer rounds, but still a logarithmic number of them. Relevant asymptotic performance measures for these approaches are provided in Table I.

C. Approximate Reconciliation Trees

Bloom filters are the preferred data structures when the working sets of the two peers have small resemblance. Bloom filters are less efficient when the resemblance is large, in which case a search-based approach would seem more appropriate computationally. We propose a new data structure, approximate reconciliation trees (ARTs), combining the compact representation of Bloom filters with the speed of a search-based approach.

ARTs [6] build upon the Merkle tree methods of Section B. As before, each peer first builds a Patricia trie of their set along with the associated Merkle tree values. The message A then sends to B is a Bloom filter of the values from the Merkle tree. B identifies an element x in the set difference by using the simple path traversal algorithm described earlier on its Patricia trie T_B. But now, B is comparing nodes in its collapsed trie against the Bloom filter representation of S_A, i.e. B checks the value of a node in T_B by performing a lookup in the Bloom filter provided by A. This tests whether *any* node in T_A has that value.

Using Bloom filters to summarize the Merkle tree values (depicted in panel (d) of Figure 3) has the following advantages:

- Fast search times associated with the search-based paradigm are preserved.
- Complications associated with collapsed Patricia tries are avoided, as no explicit bookkeeping data is transmitted.
- Hash collisions within the Merkle tree are virtually eliminated since the size of hashes in the local tree does not affect the message size.

The main disadvantage is that comparisons between nodes in the tries now correspond to Bloom filter lookups, which are less accurate than direct comparisons of Merkle tree values.

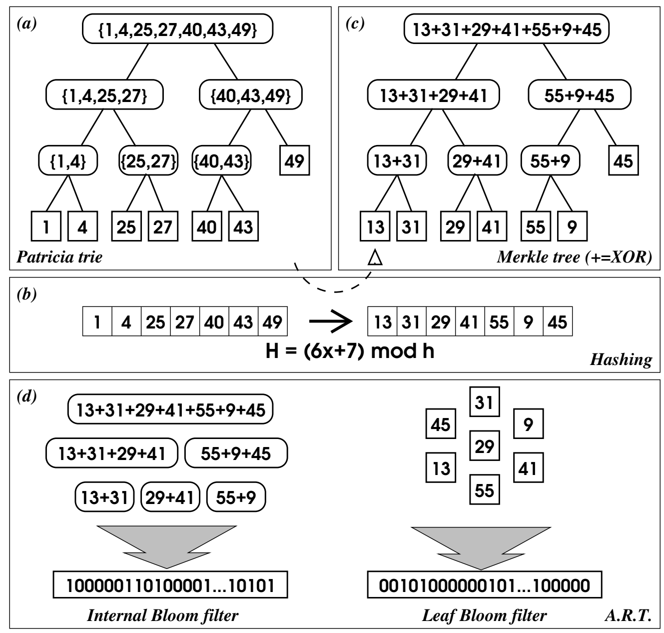


Fig. 3. Components of the ART data structure (universe size $|U| = h = 64$).

We now analyze this approach. First, because we are sending a Bloom filter of the node values, we can use a large number of bits for these values to avoid collisions ($\Theta(\log|S_B|)$ bits suffices with high probability and 64 bits covers most practical situations). We will ignore these collisions henceforth in the analysis. For B to obtain a false positive for an element x in $S_B - S_A$ at depth d in the approximation reconciliation tree for S_B, there must be a false positive for one of the d node values on the path from the root to the leaf representing x in the Bloom filter. If the false positive rate of the Bloom filter sent by A is f, the probability p_x that B identifies x as a member of $S_B - S_A$ is $p_x = (1-f)^d$. So to achieve a constant expected accuracy, f should be at most $O(1/d)$ for most elements. Since individual elements are at depth $\log|S_B| + O(1)$ in the tree of B with high probability as mentioned earlier, the false positive rate of the Bloom filter from A should be $O(1/\log|S_B|)$. This means that A should use $\Omega(\log\log|S_B|)$ bits per element in the Bloom filter. Additionally, the number of hash functions must be $\Theta(\log\log|S_B|)$ to minimize the false positive rate — using only a constant number of hash functions c requires the number of bits per element to be $\Omega\left(\sqrt[c+1]{\log|S_B|}\right)$. Our implementation also expects that a small fixed number of hash functions will be universally chosen ahead of time.

C.1 Improvements

ARTs as described so far combine some of the better properties of Bloom filters and Merkle trees, namely quicker searches for small numbers of differences without the complications of managing the tree structures. Unfortunately, they inherit a common weakness in tree-based search strategies – incorrect pruning from false positives can potentially result in large numbers of differences being overlooked. For example, if there is a false positive when checking the root of an ART, no differences will be found and the sets will be reported to be identical. Addressing this problem requires increasing the number of bits per element and the running times by non-constant factors. In this section, we discuss a series of improvements over basic ARTs to remedy these problems.

More Hashing: Our basic search method operates on the Patricia trie T_B, and hence the running time is proportional to the depth of T_B. However, the worst-case depth of the Patricia trie may still be $\Omega(|S_B|)$. To avoid this issue with high probability, we hash each element initially before inserting it into the virtual tree. The range of the hash function should be at least $\mathrm{poly}(|S_B|)$ to avoid collisions. For the analysis, we make the assumption that our hash functions are truly random. Hence we obtain a random Patricia trie, properties of which have been studied in the random search tree literature, in particular [13], [27], [15]. Specifically, the average depth of a leaf for a binary Patricia trie with n random leaf values over the interval is $\log n + O(1)$, and the variance of the depth of a leaf is constant. Moreover, the maximum depth is $\log n + \sqrt{2 \log n} + O(1)$ with high probability. Hence the distribution of the leaf depths is very closely concentrated around $\log n$.

Increased Branching Factor: The next improvement is very simple: increase the maximum branching factor of the trees. This reduces the number of internal nodes, thereby improving the false positive probability by allowing more bits per node. It also decreases the height of the tree, reducing the number of Bloom filter tests by a constant factor. The cost is a potential increase in running time, since in searches for elements of $S_B - S_A$, all children of an internal node must be checked when a match does not occur. This improvement does not change the asymptotic results, only constant factors. However, larger branching factors are an important part of a real implementation. Branching factors that are a power of two, say 2^i, are especially desirable, since then all trie operations can be performed on a fixed stride length of i bits.

Correction Levels: Our next improvement is based on the inherent redundancy in the Merkle tree structure. In the event that a match between internal nodes of the tree (identical subsets) is caused by a false positive, it is often the case that its children do not match, thereby revealing the discrepancy with certainty. Therefore we can "double-check" a match at an internal Merkle tree node, by verifying that its children match as well. The one-sided error of Bloom filters ensures that this method applies even when a filtered representation of the Merkle tree is used. As we demonstrate experimentally, this can significantly improve the accuracy at the expense of running time.

More generally, we change the search procedure so that it stops searching a path for differences only when there is a match at a node and at its c immediate ancestors, where c is a new parameter we call the *correction level*. For $c = 0$, this is exactly the same as the basic reconciliation procedure. For $c = 1$, the traversal is pruned after two consecutive matches, for $c = 2$, the traversal is pruned after three consecutive matches, and so on. If the branching factor is b, this slows down B's traversal by a factor of at most b^c.

Improved Bit Allocation: One implicit assumption in the basic ART construction is that all nodes have equal worth, since we use the same number of bits per node. But as noted earlier, false positives near the root of tree can lead to many or all of the differences being missed, an effect only partially mitigated by use of correction levels. Merely adding a few more bits to the root (and its close descendants) can significantly alleviate this.

At the opposite end of the tree, false positives at the leaves are impossible to correct. Therefore, the accuracy of ARTs is at best as accurate as the Bloom filter tests at the leaves, regardless of what is done for internal nodes. This further implies that the accuracy of an approximate reconciliation tree is no better than that of a Bloom filter of the same size. It also suggests that schemes that can reduce the number of bits used to represent an internal node without compromising the accuracy would be desirable. In fact, one can prove that a reasonable approach is to use a separate Bloom filter for each level of the tree, whereby $\Theta(h)$ bits are used to represent each node at height h in the tree.

Leveraging Random Tree Structure: A problem we have noted previously is that if we do not use a Bloom filter to represent nodes, then there may be difficulties in determining correspondences between the peers' ARTs. On the other hand, using a Bloom filter increases the probability of an individual false positive. If possible, it would be desirable to have both the higher accuracy of a Merkle tree without the complications of reconstructing the tree structure.

To enable such a hybrid, we make use of the observation that given a random Patricia trie with n nodes, the first $\log n - 2 \log \log n$ levels are complete with high probability. Since these top levels are complete, the values for these levels may simply be enumerated in some fixed order. Given this setup, B can access nodes in the first $\log |S_A| - 2 \log \log |S_A|$ levels directly, yielding a lower false positive rate than if Bloom filters were used; B then switches to testing nodes in the lower levels of the tree with a Bloom filter. Even better, since there are only $\Theta(|S_A| / \log^2 |S_A|)$ of these special values at the top of the tree, we can adopt the idea of varying numbers of bits per elements by using large hashes of $\Theta(\log |S_A|)$ bits without significantly affecting the number of bits available for the leaves and other lower levels. Specifically, this hybrid approach uses only $\Theta(|S_A| / \log |S_A|)$ bits but manages to avoid false positives in the upper levels with high probability.

C.2 Analysis

Using the various improvements of Section C.1 and incorporating the pieces of analysis for those improvements, we can now prove the following theorem.

Theorem 2: There exists a one-round protocol with message size $O(|S_A|)$, construction time $O(|S_A| \log |S_A|)$ for A, construction time $O(|S_B| \log |S_B|)$ for B, reconciliation time $O(|S_B - S_A| \log |S_B|)$ with high probability, and constant accuracy.

Proof: A constructs an ART with the following components. First, values for the top $\log|S_A| - 2\log\log|S_A|$ complete levels are explicitly sent using $\Theta(\log|S_A|)$ bits per value. Values for the remaining internal nodes are sent in a Bloom filter using $\Theta(|S_A|)$ bits and a constant number of hash functions; similarly, the leaf nodes are sent in a separate Bloom filter using $\Theta(|S_A|)$ bits and a constant number of hash functions. Without loss of generality, we may assume that $|S_B| \leq 2|S_A|$; otherwise, the leaf Bloom filter itself can be used to satisfy theorem.

In the top levels of the tree with explicit hash values, the probability of a hash collision is $1/2^{\Theta(\log|S_A|)} = 1/\text{poly}(|S_A|)$. Using a union bound, the probability that a hash collision appears anywhere along a path through the top levels occurs with probability only $O(\log|S_A|/\text{poly}(|S_A|))$. Searching through these levels takes $O(d\log|S_A|)$ time.

For the remaining levels, B uses a correction level of $\Theta(\log\log\log|S_B|) = \Theta(\log\log\log|S_A|)$. This is only sufficient to guarantee a constant probability of a false positive along a path through the first $\log|S_A| + O(\log\log|S_A|)$ levels of the tree, but this depth encompasses a constant fraction of the leaf nodes, ensuring a constant accuracy. The cost of this correction level is a slowdown of $b^{\Theta(\log\log\log|S_A|)} = \text{poly}(\log\log|S_A|)$ for any fixed branching factor b. Thus, searching the remaining levels takes $O(d * \text{poly}(\log\log|S_A|)) = O(d\log|S_A|)$ time. Combining the bounds proves the theorem. ■

VII. Experimental Results

We first investigate the effectiveness of collaboration using our methods. Then in Section B, we compare the performance and accuracy of approximate reconciliation methods.

A. Collaborative Experiments

Our collaborative experiments demonstrate the benefits and costs of using reconciliation in peer-to-peer transfers and in parallel downloads. The simple scenarios we present are designed to be illustrative and highlight the primary benefits of our methods; the performance improvements we demonstrate can be extrapolated onto more complex scenarios.

A.1 Simulation Parameters

All of our collaborative experiments consider transfers of a 128MB file. We assume that the origin server divides this file into input symbols of 1400 bytes each, to fit in an Ethernet packet with headers, and subsequently encodes this file into a large set of encoding symbols. We associate each encoding symbol with a 64bit identifier representing the set of input symbols used to produce it. The irregular degree distribution used in the codes was generated using heuristics based on the discussion in Section V and described in [12]. This degree distribution had an average degree of 11 for the encoding symbols and average decoding overhead of 2.3%. The experiments used the simplifying assumption of a constant decoding overhead $\epsilon_d = 2.5\%$. For recoding, we pre-computed degree distributions in the same fashion with all input parameters rounded to increments of 0.05 and a maximum degree of 50. Min-wise summaries used 180 permutations, yielding 180 entries of 64 bits each for a total of 1440 bytes per summary. Bloom filters used 6 hash functions and $8(1+\epsilon_d)\ell$ bits for a total of 96 KB per filter. Separate results for ARTs are omitted in this section since their overhead is visually indistinguishable from Bloom filters. Bloom filters and ARTs differ primarily in terms of accuracy and speed, the focus of Section B.

A.2 Collaboration Methods

We compare three methods of orchestrating collaboration in our experiments, described both in increasing order of complexity and performance. While our methods may be combined in other ways, these scenarios exhibit the basic tradeoffs.

Uninformed Collaboration: The sending peer picks a symbol to send at random. This is the strategy used by Swarmcast [31].

Speculative Collaboration: The sending peer uses a min-wise summary from the receiving peer to estimate the containment and heuristically tune the degree distribution of recoded symbols which it encodes and sends. The containment estimated from the min-wise summary and the number of symbols requested are used to pick a pre-generated distribution tuned as described earlier. Fractions used in picking pre-generated distributions were rounded down to multiples of 0.05 except when the desired fraction would be zero.

Reconciled Collaboration: The sending peer uses either a Bloom filter or an ART from the receiving peer to filter out duplicate symbols and sends a random permutation of the differences. Random permutations of the transmitted encoding symbols are used to minimize the likelihood that two distinct sending peers send identical sequences of encoding symbols to the receiving peer.

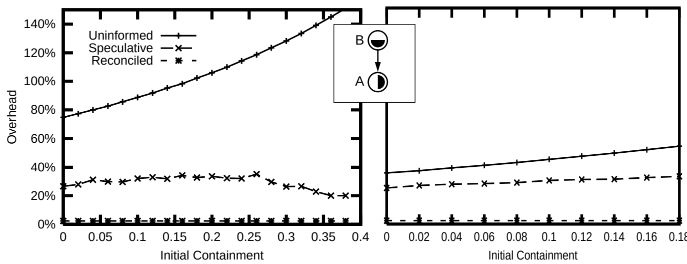
A.3 Scenarios and Evaluation

We begin by varying three experimental factors: the set of connections in the overlay formed between sources and peers, the distribution of content among collaborating peers, and the slack of the scenario, defined as follows.

Definition 3 (Slack) The *slack* s associated with a set of peers Y is $\frac{|\bigcup_{X\in Y} S_X|}{\ell}$ where S_X is the working set of peer X and ℓ is the total number of input symbols.

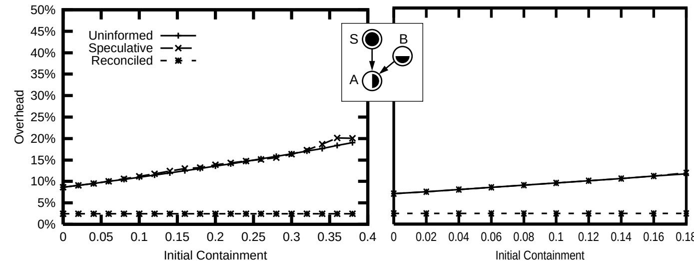
By this definition, in a scenario of slack s, there are $s\ell$ distinct encoding symbols in the working sets of peers in Y. Clearly, when the slack is less than $1+\epsilon_d$, the set of peers Y will be unable to recover the file even if they use an exact reconciliation algorithm, since the decoding overhead alone is ϵ_d. When the slack is larger than $1+\epsilon_d$, and if peers are using a reconciliation algorithm with accuracy a, then they can expect to be able to retrieve the file if $(1+\epsilon_d) \leq sa$. Our methods provide the most significant benefits over naive methods when there is only a small amount of slack; as noted earlier, approximate reconciliation is not especially difficult when the slack is large. We use slack values of 1.1 and 1.3 for comparison between compact scenarios with little available redundancy and looser scenarios.

For simplicity, we assume that each connection has the same amount of available bandwidth; our methods apply irrespective of this assumption. The receiving peer A for whom we measure the overhead always starts with 0.5ℓ encoding symbols from the server. The encoding symbols known to the sending peers are determined by the slack of the scenario and the containment



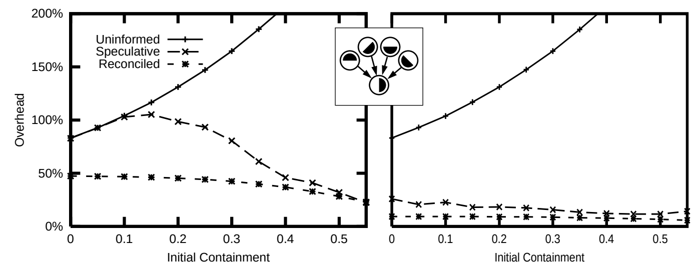
(a) Slack = 1.1 (b) Slack = 1.3

Fig. 4. Overhead of peer-to-peer reconciliation.



(a) Slack = 1.1 (b) Slack = 1.3

Fig. 5. Overhead of peer-augmented downloads



(a) Slack = 1.1, no updated summaries (b) Slack = 1.1, updated summaries (f=10)

Fig. 6. Overhead collaborating with multiple peers in parallel.

defined in Section IV.

For each technique, we measure its average overhead on a set of trials, where an average overhead of ϵ means that $(1+\epsilon)\ell$ symbols need to be received on average to recover a file of ℓ input symbols. In case of a server sending encoded content without aid from peers with partial content, the overhead is merely the decoding overhead, i.e. $\epsilon = \epsilon_d$. In other scenarios, there may be additional *reception overhead* arising from receiving duplicate encoded symbols or *recoding overhead* from receiving useless recoded symbols. The x-axis of each plot is the range of containment of the sending peers by the receiving peer. Each data point is the average of 50 simulations.

A.4 Peer-to-Peer Reconciliation

The simplest scenario, depicted in Figure 4, consists of two peers with partial content in which one peer sends symbols to the other. This scenario demonstrates the feasibility of our approach even in the worst case when servers with the original file are no longer available, and reconciliation is barely possible.

For receiving peer A, sending peer B, with a file consisting of ℓ input symbols and slack s, $\ell s = |S_A| + |S_B| - |S_A \cap S_B|$. By the definition of containment, $c = \frac{|S_A \cap S_B|}{|S_B|}$, so $|S_B| = (\ell s - |S_A|)/(1-c)$. These two equations therefore uniquely determine $|S_A|$, $|S_B|$, and $|S_A \cap S_B|$ as functions of the slack and the containment. Also, we need to constrain the scenarios to cases where neither A nor B can recover the file initially. This gives an upper bound on feasible values of c for a given slack s, and explains the variation in values on the x-axes of our plots.

Figure 4 shows the results of our experiments for this scenario. In each experiment, uninformed collaboration performs poorly and degrades significantly as the containment increases. The intuition is that the rate of useless symbols transmitted increases as the number of symbols shared between peers increases. Shared symbols increase as containment increases (and as each transfer progresses). This can be analyzed exactly as a variant of the well known Coupon Collector's problem.

Speculative collaboration is more efficient, but the overhead still increases slowly with containment. In contrast, the overhead of reconciled collaboration is virtually indistinguishable from the overhead of encoding alone and does not increase with containment. The extra overhead of reconciliation is purely from the cost of transmitting a Bloom filter or ART, or less than a percent when sending 8 bits for every symbol (1400 bytes).

A.5 Peer-Augmented Downloads

Our next scenario considers a download from a server with complete content, supplemented by a concurrent transfer from a peer, as illustrated in Figure 5. The overhead of uninformed collaboration is considerably lower than in the scenarios of Figure 4, primarily because a larger fraction of the content is sent directly via fresh symbols from the server. Using our methods, speculative collaboration performs similarly to uninformed collaboration in this scenario, as the recoding methods used are not highly optimized In all cases, reconciled collaboration has overhead slightly higher than receiving symbols directly from the server, but the transfer time is substantially reduced due to concurrent downloads.

For this scenario, it is natural to consider the *speedup* that is obtained by augmenting the download with an additional connection, i.e. the ratio between the transfer rate of the augmented method and the transfer rate using a single sender (incurring no decoding overhead), Augmenting a download with a connection of equal available bandwidth with a reconciled transfer having 0.025 overhead achieves a speedup of 1.95, while augmenting a download with an uninformed transfer having 0.20 overhead achieves a more modest speedup of 1.67.

A.6 Collaborating with Multiple Peers in Parallel

Finally, we consider a parallel download scenario similar to those of [8]. Here, a peer is collaborating concurrently with four peers, all with partial content, as pictured in Figure 6. This scenario demonstrates that given appropriate reconciliation algorithms, one can leverage bandwidth from peers with partial content with only a slight increase in overhead.

When encoding symbols are allocated across multiple peers, slack and containment no longer uniquely determine the initial distribution of symbols. We allocate the symbols as follows. As before, the receiver is initially allocated 0.5ℓ symbols. Each of these 0.5ℓ symbols is also given to each sending peer independently with probability c. The remaining symbols are given to a sending peer with probability p such that $\frac{p}{1-(1-p)^4} = \frac{(0.5/t)(1-c)}{1-(0.5/t)}$. Any of these later symbols which is not given to any peer is discarded and replaced. Each peer thus has 0.5ℓ symbols (expected) at the beginning of the experiment.

The results of this scenario are shown in Figure 6(a). As one would expect, uninformed collaboration performs extremely poorly. For low values of containment, speculative collaboration performs the same as uninformed collaboration, but dramatically improves as containment increases. In contrast to previous experiments, reconciled collaboration has much higher overhead than before. This arises from correlation across multiple peers. Sending peers D and E may identify shared symbol x as being in $S_D - S_A$ and $S_E - S_A$, respectively, and then both send x to receiving peer A. When a symbol is received multiple times, it directly contributes to the overhead. For similar reasons, the performance of speculative collaboration is also degraded, as the recoding algorithm is optimized only for transfers between pairs of peers. For lack of space, we omit results for higher slack values, but note that these scenarios are inherently easier, given their greater similarities to those of [8].

Given the poor performance of reconciled collaboration when there is sharing between sending peers, we consider the effects of periodically updating summaries. The previous experiments performed coarse-grained reconciliation only once initially. Figure 6(b) shows the results of this experiment when peers periodically exchange summaries, after receiving each new set of $\ell/10$ symbols. Here, speculative collaboration updates the min-wise summary and reconciled collaboration updates the Bloom filters, and both show dramatic improvements.

B. Reconciliation Experiments

Next, we compare Bloom filters to ARTs. Our experiments focus on settings where the speed of reconciliation is the primary concern, and the achievable accuracy is of secondary importance. Most of these experiments deal with sets of 10,000 random 32-bit elements. In one set of experiments, the number of differences is fixed at 100 differences; in another, the number of differences varies over several orders of magnitude. The total message size is 8 bits per element while the hashes used internally use 64 bits of resolution. In all experiments, each data point shown is the average of 1,000 sample runs.

B.1 Accuracy Experiments

We begin with accuracy experiments, in order to find good parameter settings and investigate basic tradeoffs. Figure 7(a) shows the results of using a basic implementation in which a binary trie is used, along with two Bloom filters: one for the internal nodes and one for the leaves. Figure 7(b) demonstrates the results of increasing the branching factor combined with the hybrid approach where the complete levels of the Merkle tree are sent explicitly. Both experiments have the same setup. On the x-axis, we vary the number of bits allocated to the leaf Bloom filters (the remaining bits of the 80,000 available are devoted to internal nodes). On the y-axis, we plot the accuracy obtained by the particular setting selected. The uppermost curve in each plot is the accuracy that can be obtained using brute-force searching through the leaf Bloom filters, i.e. foregoing the speed advantage of using the tree structure, but minimizing the incidence of false positives. The remaining curves illustrate the accuracy that can be obtained by various correction levels. In all cases, increasing the correction level c used by B brings the accuracy closer to that of the leaf Bloom filter. At the same time, as the correction level increases, the optimal distribution of bits allocates more bits to the leaves as correction reduces the internal false positive rate. For the basic binary implementation, note that using no correction delivers rather poor accuracy, but boosting the correction level moderately delivers accuracy comparable to that of Bloom filters alone. The full implementation achieves a dramatic increase in accuracy for all settings. One observation we make here is that the optimal allocation of bits between the internal node and leaf bloom filters is shifted in favor of the leaf bloom filter — this is because each of the other optimizations works to reduce the internal false positive rate.

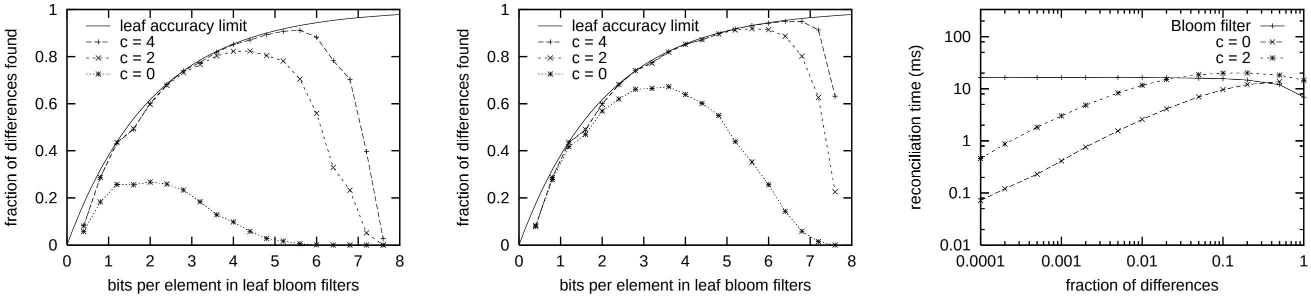
B.2 Speed Experiments

We now turn to the question of speed, running experiments on a 1 GHz Pentium 3 with 256 MB of RAM. Based on our earlier results, all of the ARTs use the hybrid approach with a branching factor of four and two Bloom filters, one for leaf nodes and one for internal nodes. For each case, we consider correction levels of zero and two, and choose the distribution of bits across the two Bloom filters to maximize accuracy. In all cases, we compare performance against vanilla Bloom filters. As depicted in Figure 7(c), we vary the number of differences over several orders of magnitude and compare reconciliation times, which also vary over several orders of magnitude. To depict such wide variability, the plot is presented on log-log scale. For small numbers of differences, vanilla Bloom filters take significantly longer to reconcile. The reconciliation time taken by Bloom filters is roughly constant as the number of differences varies, but drops slightly as the number of differences increases. (This is because in using the Bloom filter, positive queries require evaluating *all* of the hash functions, whereas negative queries may terminate once *any* hash function reveals the element is not in the set.) The reconciliation time for ARTs grows roughly linearly with the number of differences and is initially very small. As with Bloom filters, there is also a drop in the time to reconcile when nearly all the elements are different.

When using a correction level of two, ARTs are faster if the number of differences is fewer than 2% of $|S_B|$. Without correction, they are faster if the number of differences is fewer than 30% of $|S_B|$, but at the cost of significantly decreased accuracy.

VIII. Our Work in Context

Since a preliminary version of this paper appeared in conference form [7], there has been considerable work complementary to or directly relating to our findings. In particular, Bullet [17] implements many of our ideas, and adds explicit mechanisms for promoting diversity among working sets. Their work includes a full implementation and extensive experiments over the Internet and in simulation, reinforcing our findings and showing up to a factor of two improvement in throughput from using informed content delivery in a variety of situations.



(a) Accuracy of basic approach. branching factor 2, 100 differences.

(b) Accuracy of hybrid approach. branching factor 4, 100 differences.

(c) Speed of various approaches with varying numbers of differences.

Fig. 7. Reconciliation Performance. $|S_A| = |S_B| = 10000$, 8 bits per element.

LT codes [18] and online codes [21] improve upon the best published erasure codes, with the latter having the stated intent of improving collaboration methods such as ours. Unlike other methods, these codes produce encodings whose overhead is independent of the size of the source file. The work of [21] also provide ideas for reconciliation that include sequential labeling of the packets in encoding streams and exchanging sequence number ranges to reconcile. As we have argued earlier, the complexity of reconciliation using subranges of received packets grows with transience of network conditions and topology, and can perform as poorly as an enumeration-based approach in the worst case. Also, this proposal relies on in-order packet delivery via reliable connections, while ours does not.

SplitStream [10] leverages a distributed hash table topology to maintain multiple end-system multicast trees, rooted at the origin server. Each tree receives a distinct stream of encoded content. Additionally, the trees have the property that each peer is an internal node in exactly one end-system multicast tree. This system is an ideal environment for our techniques, as the overlay is richly connected, content among peers is diversified, and multiple paths are used to route content to each peer.

The PRM [2] end-system multicast protocol has a motivation similar to our work. It focuses on handling packet loss and providing high delivery rates for each packet in a session. The authors of this work augment an end-system multicast tree with a small number of random edges between nodes. In their architecture, a duplicate of each received packet is also forwarded along a random edge. Their scheme also performs localized reconciliation over a small sliding window of content.

A. *Other Suitable Applications*

Reliable delivery of large files using erasure-resilient encodings is only one representative example of content delivery scenarios that can benefit from the approaches proposed in this paper. More generally, any content delivery application which satisfies the following conditions may stand to benefit.

- The architecture employs a rich overlay topology potentially involving multiple connections per peer.
- Peers may only have a portion of the content, with potentially complex correlations among their working sets.
- Working sets of peers are drawn from a large universe of possible symbols.

An example application which satisfies these criteria is video-on-demand, which also involves reliable delivery of a large file, but with additional complications due to timeliness constraints, buffering issues, etc. Our methods can naturally be utilized in conjunction with existing approaches for video-on-demand such as [20] to move from a pure client-server model to an overlay-based model. While the methods of [20] also advocate the use of erasure-resilient codes, informed content delivery for video-on-demand can apply whether or not codes are used. A similar approach can be used for near real-time delivery of live streams. For this application, where reliability is not necessarily essential, collaboration may improve best-effort performance.

Finally, our approach may be used for peer-to-peer applications running a shared virtual environment, such as distributed interactive simulation or networked multi-player games. Here, peers may only be interested in reconstructing a small region of what can be a very large-scale environment.

IX. Conclusions

Overlay networks offer a powerful alternative to traditional mechanisms for content delivery, especially in terms of flexibility, scalability and deployability. In order to derive the full benefits of the approach, some care is needed to provide methods for representing and transmitting the content in a manner that is as flexible and scalable as the underlying capabilities of the delivery model. We argue that straightforward approaches at first appear effective, but ultimately suffer from similar scaling and coordination problems that have undermined other multipoint service models for content delivery.

In contrast, we argue that a digital fountain approach to encoding the content affords a great deal of flexibility to end-systems performing large transfers. The main drawback of the approach is that the large space of possible symbols in the system means that coordination across end-systems is also needed here, in this case to filter useful content from redundant content. Our main contributions furnish efficient, concise representations which sketch the relevant state at an end-system in a handful of packets and then provide appropriate algorithmic tools to perform well under any circumstances. With these methods in hand, informed and effective collaboration between end-systems can be achieved, with all of the benefits of using an encoded content representation.

REFERENCES

[1] ANDERSEN, D., BALAKRISHNAN, H., KAASHOEK, F., AND MORRIS, R. Resilient overlay networks. In *Proc. of ACM SOSP* (Banff, Canada, October 2001).

[2] BANERJEE, S., LEE, S., BHATTACHARJEE, B., AND SRINIVASAN, A. Resilient multicast using overlays. In *Proc. of ACM Sigmetrics* (June 2003).

[3] BLOOM, B. Space/time trade-offs in hash coding with allowable errors. *Communications of the ACM 13* (July 1970), 422–426.

[4] BRODER, A. On the resemblance and containment of documents. In *Compression and Complexity of Sequences (SEQUENCES)* (Positano, Italy, June 1997).

[5] BRODER, A. Z., CHARIKAR, M., FRIEZE, A. M., AND MITZENMACHER, M. Min-wise independent permutations. *Journal of Computer and System Sciences 60*, 3 (2000), 630–659.

[6] BYERS, J. W., CONSIDINE, J., AND MITZENMACHER, M. Fast approximate reconciliation of set differences. Tech. Rep. BUCS-TR-2002-019, Boston University, July 2002.

[7] BYERS, J. W., CONSIDINE, J., MITZENMACHER, M., AND ROST, S. Informed content delivery across adaptive overlay networks. In *Proc. of ACM SIGCOMM* (Pittsburgh, PA, August 2002), pp. 47–60.

[8] BYERS, J. W., LUBY, M., AND MITZENMACHER, M. Accessing multiple mirror sites in parallel: Using Tornado codes to speed up downloads. In *Proc. of IEEE INFOCOM* (March 1999), pp. 275–83.

[9] BYERS, J. W., LUBY, M., AND MITZENMACHER, M. A digital fountain approach to asynchronous reliable multicast. *IEEE Journal on Selected Areas in Communications 20(8)* (October 2002), 1528–40.

[10] CASTRO, M., DRUSCHEL, P., KERMARREC, A.-M., NANDI, A., ROWSTRON, A., AND SINGH, A. Splitstream : High-bandwidth content distribution in a cooperative environment. In *Proceedings of 2nd International Workshop on Peer-to-Peer Systems* (February 2003).

[11] CHU, Y.-H., RAO, S., SESHAN, S., AND ZHANG, H. A case for end system multicast. *IEEE Journal on Selected Areas in Communications 20(8)* (October 2002), 1456–71.

[12] CONSIDINE, J. Generating good degree distributions for sparse parity check codes using oracles. Tech. Rep. BUCS-TR 2001-019, Boston University, October 2001.

[13] DEVROYE, L. A note on the probabilistic analysis of Patricia tries. *Random Structures and Algorithms 3* (1992), 203–214.

[14] JANNOTTI, J., GIFFORD, D., JOHNSON, K., KAASHOEK, M., AND O'TOOLE, J. Overcast: Reliable multicasting with an overlay network. In *Proc. of USENIX Symp. on Operating Systems Design and Implementation)* (San Diego, CA, October 2000).

[15] KNESSL, C., AND SZPANKOWSKI, W. Limit laws for heights in generalized tries and PATRICIA tries. In *Proc. LATIN 2000* (2000).

[16] KNUTH, D. E. *The Art of Computer Programming, Volume 3: Sorting and Searching*. Addison-Wesley, Reading, MA, 1973.

[17] KOSTIC, D., RODRIGUEZ, A., ALBRECHT, J., AND VAHDAT, A. Bullet: high bandwidth data dissemination using an overlay mesh. In *Proc. of ACM SOSP* (2003), ACM Press, pp. 282–297.

[18] LUBY, M. LT Codes. In *Proceedings of 43rd Symposium on Foundations of Computer Science (FOCS 2002)* (Vancouver, November 2002).

[19] LUBY, M., MITZENMACHER, M., SHOKROLLAHI, A., AND SPIELMAN, D. Efficient erasure correcting codes. *IEEE Transactions on Information Theory 47(2)* (2001), 569–584.

[20] MAHANTI, A., EAGER, D. L., VERNON, M. K., AND SUNDARAM-STUKEL, D. Scalable on-demand media streaming with packet loss recovery. *IEEE/ACM Transactions on Networking 11(2)* (2003), 195–209.

[21] MAYMOUNKOV, P., AND MAZIERES, D. Rateless codes and big downloads. In *Proceedings of 2nd International Workshop on Peer-to-Peer Systems* (February 2003).

[22] MERKLE, R. A digital signature based on a conventional encryption function. In *CRYPTO* (Santa Barbara, CA, August 1987).

[23] MINSKY, Y., AND TRACHTENBERG, A. Efficient reconciliation of unordered databases. Tech. Rep. TR1999-1778, Cornell University, 1999.

[24] MINSKY, Y., AND TRACHTENBERG, A. Scalable set reconciliation. In *Proceedings of 40th Annual Allerton Conference on Communication, Control, and Computing* (October 2002).

[25] MINSKY, Y., TRACHTENBERG, A., AND ZIPPEL, R. Set reconciliation with nearly optimal communication complexity. In *Proc. of IEEE Int'l Symp. on Information Theory* (Washington, DC, June 2001).

[26] MITZENMACHER, M. Compressed Bloom Filters. *IEEE/ACM Transactions on Networking 10(5)* (October 2002), 604–612.

[27] PITTEL, B., AND RUBIN, H. How many random questions are necessary to identify n distinct objects? *J. of Combinatorial Theory, Ser. A 55* (1990), 292–312.

[28] RABIN, M. Efficient dispersal of information for security, load balancing and fault tolerance. *Journal of the ACM 38* (1989), 335–348.

[29] RODRIGUEZ, P., AND BIERSACK, E. W. Dynamic parallel-access to replicated content in the Internet. *IEEE/ACM Transactions on Networking 10(4)* (August 2002).

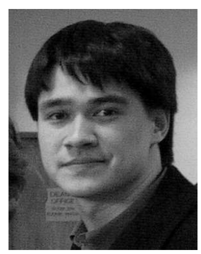
[30] SAVAGE, S., COLLINS, A., HOFFMAN, E., SNELL, J., AND ANDERSON, T. The end-to-end effects of Internet path selection. In *Proc. of ACM SIGCOMM* (August 1999).

[31] Swarmcast. http://www.opencola.org/projects/swarmcast.

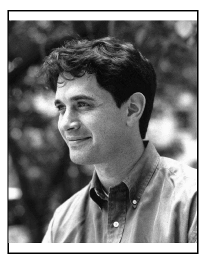
[32] VITTER, J. S. Random sampling with a reservoir. *ACM Trans. on Math. Software 11* (1985), 37–57.

John W. Byers is Assistant Professor of Computer Science at Boston University and Affiliated Scientist at Digital Fountain, Inc. Prior to joining B.U., he completed his Ph.D. in Computer Science at the University of California at Berkeley in 1997 and was a postdoctoral researcher at the International Computer Science Institute in Berkeley in 1998. His research interests include algorithmic aspects of networking, Internet content delivery, and network measurement.

Dr. Byers received a National Science Foundation CAREER Award in 2001. He serves on the program committees for numerous conferences, including ACM SIGCOMM, ACM SIGMETRICS and IEEE ICNP. He is currently on the editorial board of IEEE/ACM Transactions on Networking, and has been a member of the ACM since 1999.



Jeffrey Considine received his BA and MA degrees in computer science at Boston University in 1999. He is a Ph.D. candidate in computer science at Boston University, where he does research on randomized data structures and overlay networking.



Michael Mitzenmacher (M '01 / ACM M '99) received his Ph.D. degree from the University of California at Berkeley in 1996. He was a Research Scientist with the Digital Systems Research Center, Palo Alto, CA, from 1996 to 1998. In 1999, he joined the faculty at Harvard University, where he has been an Associate Professor since 2002. He is the coninventor on twelve issued patents and has written over ninety conference and journal publications. His research interests include the design and analysis of algorithms, dynamic processes, load balancing, Web algorithms, compression, error-correcting codes, and computer science education.

Dr. Mitzenmacher has received the NSF CAREER award and an Alfred P. Sloan Research Fellowship. In 2002 he shared in the IEEE Information Theory Society Best Paper Award for his work in error-correcting codes.

Stanislav Rost received his BA and MA degrees from the computer science department at Boston University in 2001. He is a Ph.D. candidate in computer science at MIT, where he does research on routing, content delivery and distributed computing in power-constrained wireless networks.