Informed Content Delivery Across Adaptive Overlay Networks *

John Byers
byers@cs.bu.edu

Jeffrey Considine
jconsidi@cs.bu.edu

Dept. of Computer Science
Boston University
Boston, Massachusetts

Michael Mitzenmacher
michaelm@eecs.harvard.edu

EECS
Harvard University
Cambridge, Massachusetts

Stanislav Rost
stanrost@lcs.mit.edu

MIT Laboratory for
Computer Science
Cambridge, Massachusetts

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 encoded 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 encoded 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 estimation, 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.

*John Byers, Jeffrey Considine and Stanislav Rost were supported in part by NSF grants ANI-0093296 and ANI-9986397. Michael Mitzenmacher was supported in part by NSF grants CCR-9983832, CCR-0118701, CCR-0121154, and an Alfred P. Sloan Research Fellowship.

SIGCOMM'02, August 19-23, 2002, Pittsburgh, Pennsylvania, USA.

Categories and Subject Descriptors

C.2 **[Computer Systems Organization]**: Computer-Communication Networks

General Terms

Algorithms, Measurement, Performance

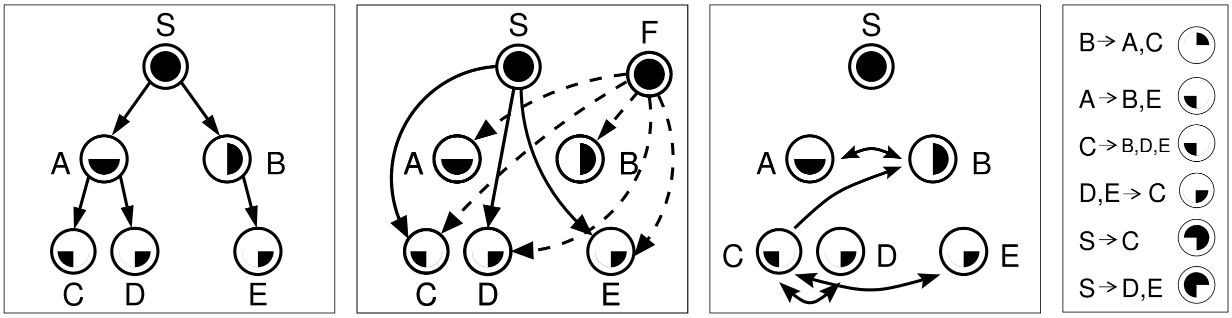
Keywords

Overlay, peer-to-peer, content delivery, digital fountain, erasure correcting code, min-wise summary, Bloom filter, reconciliation, collaboration.

1 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. First, the bandwidth consumption of such an approach is wasteful. Second, the rate of each individual transfer is limited by the characteristics of the end-to-end path between the source and that destination. The problem of excessive bandwidth consumption can be solved by a reliable multicast-based approach. With multicast, only a single copy of each packet payload transmitted by the server traverses each link in the multicast tree en route to the set of clients. Providing reliability poses additional challenges, but one elegant and scalable solution is the digital fountain approach [8], whereby the content is first encoded via an erasure-resilient encoding [18, 24, 17], 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.

Although multicast-based dissemination offers near-optimal scaling properties in bandwidth and server load, IP multicast suffers from limited deployment. This lack of deployment has led to the development of *end-system* approaches [10, 13, 9], along with a wide variety of related schemes relevant to peer-to-peer content delivery architectures [25, 27, 29, 30, 32]. Many of these architectures overcome the deployment hurdle faced by IP multicast by requiring no changes to routers nor additional router functionality. Instead, these architectures construct *overlay* topologies, comprising collections of unicast connections between end-systems, in which each edge



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

Figure 1: **Possibilities for content delivery.** Shaded content within a node in the topology represents the working set of that node. Connections in (b) supplement (a); connections in (c) supplement (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.

(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 [2, 28]. Finally, end-systems are now explicitly required to cooperate. 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 have the ability to actively collaborate, in an informed manner.

We now return to the second limitation with traditional service models based on tree topologies: the transfer rate to a client in such a topology is bounded by the available bandwidth of the bottleneck link on 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* [7], 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, informal evidence of availability of bandwidth for opportunistic downloads between collaborating peers. The legend of 1(d) depicts the portions of content which can be beneficially exchanged via opportunistic transfers between pairs of end-systems in this scenario.

As discussed earlier, the tree and directed acyclic graph topologies of Figures 1(a) and 1(b) impede the full flow of content 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 a more careful level of 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 which packets lie in the set difference of their working sets, and subsequently make an informed transfer of those packets, i.e. they must reconcile the two 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 2. In that section, we also elaborate the numerous benefits of using encoded content. The main drawback of the added flexibility provided by the use of erasure-resilient content is that reconciliation becomes a more challenging problem. To address this challenge, in Sections 3, 4 and 5, 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 6, 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 provide a recap of our results and draw conclusions in Section 7.

2 Content Delivery Across Overlay Networks

We motivate our approach first by sketching fundamental challenges that must be addressed by any content delivery architecture and outlining the set of opportunities that an overlay approach affords. Next, we argue the pros and cons of encoded content, the cons primarily being a small amount of added complexity and the pros being greatly improved flexibility and scalability. We outline the encoding building blocks we use and enumerate the benefits they provide and the costs they incur.

2.1 Challenges and Opportunities

When operating in the context of the fluid environment of the Internet, there are a number of fundamental problems 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.

Overlay networks should tolerate asynchrony and heterogeneity and should adapt to transient behavior, all in a scalable manner. For example, a robust overlay network should have the ability to adaptively detect and avoid congested or temporarily unstable [15, 2] areas of the network. Continuous reconfiguration of virtual topology by overlay management strives to establish paths with the most desirable end-to-end characteristics. While optimal paths may be difficult to identify, an overlay node can often identify paths that are better than default Internet paths [2, 28]. Such reactive behavior of the virtual topology may frequently force the nodes to reconnect to better-suited peers. But of course this adaptive behavior then exacerbates the fundamental 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. As the transfers progress, and different end-systems peer with one another, working sets will become further divergent and fragmented. By carefully orchestrating connections, one may be able to manage the level of fragmentation, but only at the expense of restricting potential peering arrangements, thereby limiting throughput.

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. Or 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.

2.2 Limitations of Stateful Solutions

Solutions to the problems and concerns of the preceding subsection cannot be scalably achieved with techniques that require state to be stored at connection endpoints. For example, 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 also problematic, as discussed in [7]. The natural approach is to divide the range of the missing packets into disjoint sets in order to download different ranges from different sources. With heterogeneous bandwidth and transient network conditions, effectively predicting the correct distribution of ranges among sources is difficult, and hence frequent renegotiation may be required. Also, there is a natural bottleneck that arises from the need to obtain "the last few packets." If an end-system has negotiated with multiple sources to obtain certain packet ranges, and one source is slow in sending the last necessary packets, the end-system must either wait or pursue a fine-grained renegotiation with other sources. 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 [26]), the use of encoding facilitates simpler and more effective parallel downloading.

One other complication is that in order to maximize the advantage of obtaining useful content from multiple peers, 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 naturally arise due to factors such as uncorrelated losses, bandwidth differences, asynchronous joins, and topology reconfigurations. But 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.

2.3 Benefits of Encoded Content

An alternative to using stateful solutions as described above is the use of the digital fountain paradigm [8] running over an unreliable transport protocol. The digital fountain approach was originally designed for point-to-multipoint transmission of large files over lossy channels. In this application, scalability and resilience to packet loss is achieved by using an *erasure correcting code* [17, 18, 24] to produce an unbounded stream of encoding symbols derived from the source file. The encoding stream has the guarantee that a receiver is virtually 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. Some implementations are capable of efficiently reconstructing the file having received only a few percent more than the number of symbols in the original file [17, 16, 8], and we assume such an implementation is used. A digital fountain approach provides a number of important benefits which are useful in a number of content delivery scenarios [8, 7, 16].

- **Continuous Encoding:** Senders with a complete copy of a file may continuously produce encoded symbols from the content.
- **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 5.4. 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, in which no state need be transferred among hosts and no dangling requests for retransmission need be resolved. 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, unordered* universe, end-systems that hold only part of the content must take care to arrange transmission of useful information between each other. 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 new encoded content at will. It does not solve this problem when an end-system can only forward encoding packets, since the receiving end-system may already have obtained those packets. To avoid redundant transmissions in such scenarios, we describe mechanisms for estimating and reconciling differences between working sets and subsequently performing informed transfers.

2.4 Suitable Applications

Reliable delivery of large files leveraging 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 correlation between the working sets of peers.
- Working sets of peers are drawn from a large universe of possible symbols.

Another natural application which satisfies these criteria is video-on-demand. This application also involves reliable delivery of a large file, but with additional complications due to timeliness constraints, buffering issues, etc. Our methods for informed content delivery can naturally be utilized in conjunction with existing approaches for video-on-demand such as [19] to move from a pure client-server model to an overlay-based model. While the methods of [19] also advocate the use of erasure-resilient codes, our methods for informed content delivery for video-on-demand apply whether or not codes are used. Similarly, informed content delivery 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 relying on a shared virtual environment, such as distributed interactive simulation or networked multi-player games. For these applications, peers may only be interested in reconstructing a small subspace of what can be a very large-scale environment. Here, in addition to issues of scalable naming and indexing, summarization is also essential for facilitating effective collaboration between peers.

3 Reconciliation and Informed Delivery

The preceding sections have established expectations for informed collaboration: an adaptive overlay architecture designed for scalable transmission of rich content. We abstract our solutions away from the issues of optimizing the layout of the overlay over time [10, 13, 2], as well as distributed naming and indexing [25, 29, 27]; our system supplements any set of solutions employed to address these issues.

The approaches to reconciliation which we wish to address 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 working set *sketches*, obtained by random sampling or min-wise sketches. Coarse approaches are not resource-intensive and allow us to estimate 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. This process can be fine-tuned using results of coarse-grained reconciliation.

Fine-grained reconciliation employs compact, searchable working set summaries such as Bloom filters or approximate reconciliation trees. 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 a modest amount of space at the end-system and a gigabyte of content will typically require a summary on the order of 1MB in size. Finally, recoded content optimizes transfers by tuning, or personalizing, the content across a particular peer-to-peer connection based on information presented in sketches. We describe these methods and their performance tradeoffs in the following sections.

4 Estimating Working Set Similarity

In this section, we present simple and quick methods for estimating 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. We also use the resemblance to optimize our recoding strategy described in Section 5.4. Since it is essential that the data to compute the resemblance be obtained as quickly as possible, our methods are designed to give accurate answers when only a single 1KB packet of data is transferred between peers. We emphasize that there are different tradeoffs involved in each of the approaches we describe; the best choice may depend on specifics of the application.

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) *The* containment *of B in A is the quantity* $\frac{|S_A \cap S_B|}{|S_B|}$.

Definition 2 (Resemblance) *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 [5] and were applied to determine the similarity of documents in search engines [1]. 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 [31]. 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 [5], 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.

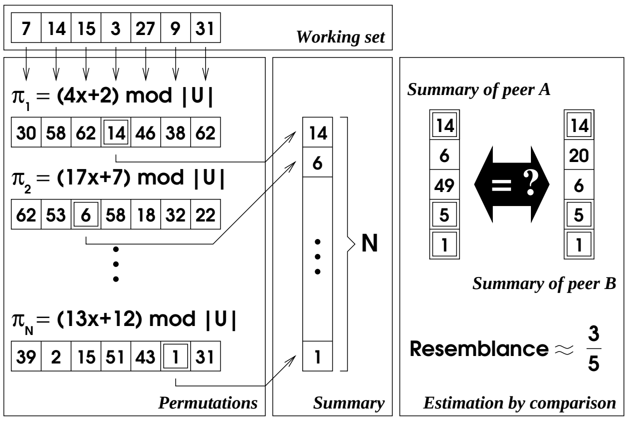


Figure 2: Example of minwise summarization and estimation of resemblance (Key universe size is 64, example permutation functions shown).

Another clever sampling technique from [5] avoids the drawbacks of the first two approaches. This approach, which we employ, calculates working set resemblance based on *min-wise sketches*, following [5, 6]; 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. Now to estimate the resemblance, peer A computes $\min \pi_j(S_A)$ for some fixed number of permutations π_j (as shown on Figure 2), 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 [4, 6].

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.

5 Reconciling Differences

As shown in the previous section, a single packet can allow peers to estimate the resemblance in their working sets. If the difference is sufficient to allow useful exchange of data, the peers may then act to determine what data to exchange. We provide methods for this problem that generally require transmission of only a handful of packets. There are a number of related performance considerations that we develop below.

The problem we consider is a set difference problem. Specifically, 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.

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 3/4 of the symbols necessary to reconstruct the file with no overlap between them. Hence we do not need exact reconciliation of the set difference; approximations will suffice. One of our contributions is this insight that *approximate* reconciliation of the set differences is sufficient and allows us to determine a large portion of $S_B - S_A$ with very little communication overhead.

In this section, we describe how to quickly determine approximate differences using Bloom filters [3]. We also introduce a new data structure, which we call an approximate reconciliation tree. Approximate reconciliation trees are especially useful when the set difference is small but still potentially worthwhile.

There are several performance considerations in designing these data structures:

- Transmission size of the message (data structure).
- Computation time.
- Accuracy of the approximation (defined below).

Definition 3 (Accuracy) *A method for set reconciliation has accuracy a if it can identify a given discrepancy between the sets of two peers with probability a.*

Traditional approaches which we will describe briefly in Section 5.1 provide perfect accuracy (i.e. accuracy equal to 1) but are prohibitive in either computation time or transmission size. Bloom filters and approximate reconciliation trees trade off accuracy against transmission size and computation time and will be described in Sections 5.2 and 5.3.

5.1 Exact Approaches

To compute differences exactly, peer A can obviously send the entire working set S_A, but 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.

Another approach is to use set discrepancy methods of [22]. If the discrepancy $d = |S_B - S_A| + |S_A - S_B|$ is known, then peer A can send a data collection of size only $O(d \log u)$ bits, or if hashing is done as pre-processing, of size only $O(d \log h)$ bits. However, if d is not known, a reasonable upper bound on d must be determined through multiple rounds of communication. In the special case where $S_A \subseteq S_B$, this information is used to find coefficients of a characteristic polynomial which is factored to recover the differences. Otherwise, a rational polynomial is interpolated and factored to recover the difference. In either case, the amount of work is

$\Theta(d^3)$. This protocol was later improved in [21] to run in expected $O(d)$ time at the cost of requiring more rounds of communication. For our application, multiple rounds of communication are undesirable, since the duration of each round is at least one round-trip time.

5.2 A Bloom Filter Approach

In our applications, it is sufficient for peer B to be able to find *most* or even just *some* of the elements in $|S_B - S_A|$. We describe how to use Bloom filters in this case.

We first review the Bloom filter data structure [3]. More details and other applications can be found in [12]. 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*, where it suggests that an element x is in S even though it is not. 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 according to the following equation: $f = (1 - \mathrm{e}^{-kn/m})^k$.

For an approximate reconciliation solution, peer A sends a Bloom filter F_A of S_A; peer B would then check for each element of S_B in F_A. When a false positive occurs, peer B assumes that peer A has a symbol that it does not have, and so peer B fails to send a symbol that would have been useful. However, the Bloom filter does not cause peer B to ever mistakenly send peer A a symbol that is not useful. As we have argued, if the set difference is large, the failure to send some useful symbols is not a significant problem.

The number of bits per element can be kept small while still achieving high accuracy. For example, using just four bits per element 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%. Using four bits per element, we can create filters for 10,000 symbols using just 40,000 bits, which can fit into five 1 KB packets. 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 [23]. For simplicity, we use only standard Bloom filters in the experiments in this paper. For computation time, $O(|S_A|)$ preprocessing is required to set up the Bloom filter, and $O(|S_B|)$ work is required to find the set difference.

The requirement for $O(|S_A|)$ preprocessing time and $O(|S_A|)$ bits to be sent may seem excessive for large $|S_A|$, especially when far fewer than $|S_A|$ packets will be sent along a given connection. There are several possibilities for scaling this approach up to larger numbers of packets. 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.

5.3 Approximate Reconciliation Trees

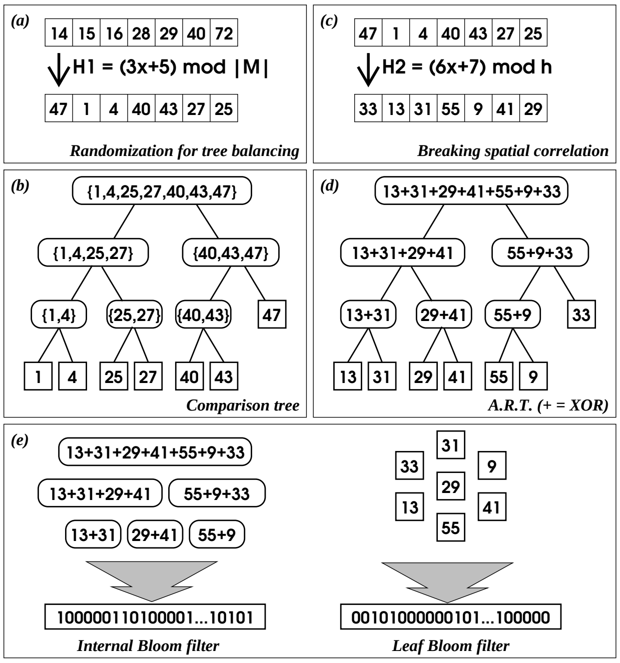


Figure 3: Example of creation and Bloom filtering of an approximate reconciliation tree. (M is $O(\text{poly } |S_A|)$; in this case, $M = |S_A|^2 = 49$, h is 64, and example permutation functions are as shown.

Bloom filters are the preferred data structures when the working sets of the two peers have small resemblance. However, our overlay approach can be useful even when the resemblance is large, and less than 1% of the symbols at peer B might be useful to peer A (this difference may still be hundreds of symbols). For this case we suggest a potentially faster approach, using a new data structure we have developed called approximate reconciliation trees.

Our approximate reconciliation trees use Bloom filters on top of a tree structure that is similar in spirit to Merkle trees, which are used in cryptographic settings to minimize the amount of data transmitted for verification [20]. We limit ourselves here to an introductory description focused on our applications here; other useful properties and applications will be detailed in a subsequent paper.

Our tree structure is most easily understood by considering the following construction. Peer A (implicitly) constructs a binary tree of depth $\log u$. The root corresponds to the whole working set S_A. The children correspond to the subsets of S_A in each half of U; that is, the left child is $S_A \cap [0, u/2 - 1]$ and the right child is $S_A \cap [u/2, u - 1]$. The rest of the tree is similar; the jth child at depth k corresponds to the set $S_A \cap [(j-1) \cdot u/2^k, j \cdot u/2^k - 1]$. Similarly, peer B constructs a similar tree for elements in S_B. Now suppose nodes in the tree can be compared in constant time, and peer A sends its tree to peer B. If the root of peer A matches the root of peer B, then there are no differences between the sets. Otherwise, there is a discrepancy. Peer B then recursively considers the children of the root. If $x \in S_B - S_A$, eventually peer B determines that the leaf corresponding to x in its tree is not in the tree for peer A. Hence peer B can find any $x \in S_B - S_A$. The total work for peer B to find all of $S_B - S_A$ is $O(d \log u)$, since each discrepancy may cause peer B to trace a path of depth $\log u$.

The above tree has $\Theta(u)$ nodes and depth $\Theta(\log u)$, which is unsuitable when the universe is large. However, almost all the nodes in the tree correspond to the same sets. In fact there are only $O(|S_A|)$ non-trivial nodes. The tree can be collapsed by removing edges between nodes that correspond to the same set, leaving only $\Theta(|S_A|)$ nodes. Unfortunately, the worst-case depth may still be $\Omega(|S_A|)$. To solve this problem we hash each element initially before inserting it into the virtual tree, as shown in Figure 3(a,b). The range of the hash function should be at least poly$(|S_A|)$ to avoid collisions. We assume that this hash function appears random, so that for any set of values, the resulting hash values appear random. In this case, the depth of the collapsed tree can easily be shown to be $\Theta(\log |S_A|)$ with high probability. This collapsed tree is what is actually maintained by peers A and B.

As seen in Figure 3(b), each node can represent a set of $\Theta(n)$ elements, which would make comparing nodes in constant time difficult. We solve this problem again with hashing, so that each set of elements corresponds to a value. The hash associated with each internal node of the tree is the XOR of the values of its children, as shown in Figure 3(d). Unfortunately, the high order bits of the first hash values of adjacent leaves in the tree are highly correlated, since this first hash determines placement in the tree. Therefore, we hash each leaf element again into a universe $U' = [1, h)$ to avoid this correlation. It is these second hash values that are used when computing the XOR of hashes in a bottom-up fashion up the tree. Checking if two nodes are equal can be done in constant time by checking the associated values, with a small chance of a false positive due to the hashing. As with Bloom filters, false positives may cause peer B to miss some nodes in the set difference $S_B - S_A$.

The advantage of the tree over a Bloom filter is that it allows for faster search of elements in the difference, when the difference is small; the time is $O(d \log |S_B|)$ using the tree instead of $O(|S_B|)$ for the Bloom filter. To avoid some space overhead in sending an explicit representation of the tree, we instead summarize the hashes of the tree in a Bloom filter. For peer B to see if a node is matched by an appropriate node from peer A, peer B can simply check the Bloom filter for the corresponding hash. This use of a Bloom filter introduces false positives but allows a small constant number of bits per element to be used while maintaining reasonable accuracy.

A false positive from the Bloom filter prematurely cuts off the search for elements in the difference $S_B - S_A$ along a path in the tree. If the false positive rate is high, the searching algorithm may never follow a path completely to the leaf. We can ameliorate this weakness by not terminating a search at the first match between nodes. Instead, we add a correction level c corresponding to the number of consecutive matches allowed without pruning the search, i.e. setting $c = 0$ terminates the search at the first match found, while setting $c = 1$ terminates the search only when matches are identified both at an internal node and a child of that node, and so on. If the correction level is greater than d, then any node in the bottom d levels of the tree is at greater risk of leading to a false positive. To cope with this problem, we use separate Bloom filters for internal hashes and leaf hashes, giving finer control over the overall false positive probability.

Figure 4 shows the results of experiments using approximate reconciliation trees. These experiments used sets of $10,000$ elements with 100 differences. For larger sets, keeping the bits per element constant will cause the error rate to increase slowly due to the tree traversals - we note that only $\Omega(\log \log \ell)$ bits per element are needed to avoid this for ℓ elements. Figure 4(a) demonstrates both the tradeoff involved when changing the number of bits used for the internal nodes and leaves while keeping the total constant and the benefits of using more levels of correction. The figure shows that using more correction levels and protecting the leaf hashes with a large number of bits per element significantly improves the fraction of differences found. For example, at $c = 5$, internal nodes are well protected against false positives, so the best performance is achieved when nearly 6 of the 8 available bits per element are allocated to the leaf filters.

Table 4(b) shows the accuracy for various numbers of bits per element and levels of correction using the optimal distribution of bits between the filters for leaves and interior nodes. The accuracy is roughly 62% when using 4 bits per element and over 90% with 8 bits per element.

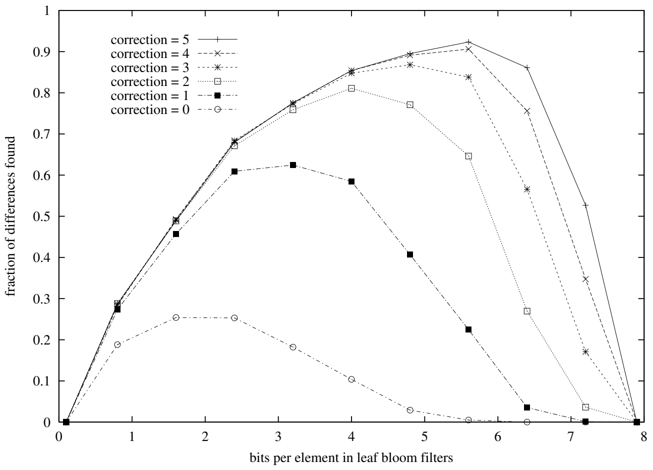
Finally, the main tradeoffs between optimized Bloom filters and approximate reconciliation trees are presented in Figure 4(c). With 8 bits per element, both data structures have over 90% accuracy, but the search time on the Bloom filter scales linearly with the size of the set, not the set difference.

5.4 Recoded Content

The final technique we describe is *recoding*, a technique which can be applied only when encoded content is employed (sketches and approximate reconciliation methods can be employed whether or not erasure correcting codes are used). Recoding is best applied when collaborating peers are known to have correlated working sets but do not yet know what elements are shared, i.e. in conjunction with coarse-grained reconciliation. One obvious possibility is for peers to send random encoding symbols, but this leads to a large amount of useless data being transmitted in many circumstances. For example, if the containment of B in A is 0.8, then sending a random symbol will be useless 80% of the time. On the other hand, as we explain more clearly below, sending a combination (using XOR) of 9 distinct output symbols is useless with probability only $0.8^9 \approx 14\%$. To describe recoding, we begin by providing relevant details for erasure correcting codes in Section 5.4.1. We then introduce recoding functionality in Section 5.4.2.

5.4.1 Sparse Parity Check Codes

To describe the recoding techniques we employ, we must first provide some additional details and terminology of sparse parity-check codes now advocated for error-correction and erasure resilience, and used in constructions which approximate an idealized digital fountain. Detailed performance evaluation of these codes in networking applications is detailed in [8]. A piece of content is divided into a collection of ℓ fixed-length blocks $x_1, \ldots, x_\ell$, each of size suitable for packetization. For convenience, we refer to these as input symbols. An encoder produces a potentially unbounded sequence of output symbols, or encoding packets, $y_1, y_2, \ldots$ from the set of input symbols. With parity-check codes, each symbol is



(a) Accuracy tradeoffs at 8 bits per element

Correction	Bits per Element			
	2	4	6	8
0	0.0000	0.0087	0.0997	0.2540
1	0.0063	0.1615	0.3950	0.6246
2	0.0530	0.3492	0.6243	0.8109
3	0.1323	0.4800	0.7424	0.8679
4	0.2029	0.5538	0.7966	0.9061
5	0.2677	0.6165	0.8239	0.9234

(b) Accuracy of approximate reconciliation trees

Data Structure	Size in bits	Accuracy	Speed
Bloom filters	$8\|S_A\|$	98%	$O(\|S_A\|)$
A.R.T. (c = 5)	$8\|S_A\|$	92%	$O(d \log \|S_A\|)$

(c) Comparison of data structures at 8 bits per element

Figure 4: Approximate Reconciliation Statistics

simply the bitwise XOR of a specific subset of the input symbols. A decoder attempts to recover the 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. The time to produce an encoding symbols from a set of input symbols is proportional to the degree of the symbol, while decoding from a sequence of symbols takes time proportional to the total degree of the symbols in the sequence, using the substitution rule defined in [17]. 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.

Provably good degree distributions have been developed and analyzed in [17, 16]. Our experience has been that heuristic approaches to generate degree distributions that leverage ideas from these works also perform well in practice for our application [11].

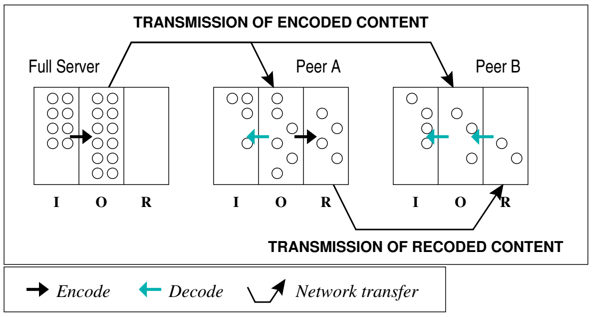
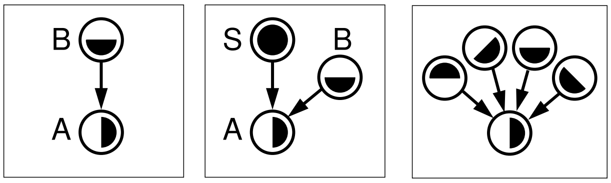


Figure 5: Example of transmission of encoded an recoded content. **I**: input symbols (original data blocks), **O**: encoded output symbols, **R**: recoded output symbols.

5.4.2 *Recoding Methods*

A recoded symbol is simply the bitwise XOR of a set of encoded symbols. Like a regular encoded symbol, a recoded symbol must be accompanied by a specification of the symbols blended to create it. To specify the input symbols combined, a recoded symbol must also list identifiers for the encoded symbols from which it was produced. As with normal sparse parity check codes, irregular degree distributions work well, although we advocate use of a fixed degree limit primarily to keep the listing of identifiers short. Encoding and decoding are performed in a fashion analogous to the substitution rule. For example, a peer with output symbols y_5, y_8 and y_{13} can generate recoded 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 output symbols are recovered, the normal decoding process proceeds. The overall flow from input symbols to recoded symbols and back in an example where a server is directly connected to two peers and the two peers are engaged in an additional collaboration is illustrated in Figure 5.

To get a feel for the probabilities involved, we consider the probability that a recoded symbol is immediately useful. Assume peer B is generating recoded symbols from file F for peer A and by virtue of a transmitted sketch, knows the containment $c = \frac{|S_A \cap S_B|}{|S_B|}$. The probability that a recoded symbol of degree d immediately yields a new encoded symbol is $\frac{\binom{c|S_B|}{d-1}\binom{(1-c)|S_B|}{1}}{\binom{|S_B|}{d}}$. This is maximized for $d = \left\lceil \frac{c}{1-c} \right\rceil$. (Note that as recoded 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 recoded symbol of this degree runs a large risk of being useless. Thus we use this value of d as a lower limit on the actual degree generated, and generate degrees between this value and the maximum allowable degree, inclusively. Recoded symbols which are not immediately useful are often eventually useful with the aid of recoded (or encoded) symbols which arrive later. By increasing the degree at the cost of im-



(a) (b) (c)

Figure 6: Scenarios considered in our experiments. (a) Peer-to-peer reconciliation, (b) Peer-to-peer collaboration augmenting a download, (c) Download from multiple peers in parallel.

mediate benefit, the probability of completely redundant symbols is substantially reduced.

6 Experimental Results

Our experiments focus on showing the overhead and potential speedups of using our methods in peer-to-peer reconciliation as well as in the setting of downloads augmented by collaborative transfers. We first show the feasibility of reconciling with a peer with partial content, by demonstrating the overhead in receiving symbols from such a sender. Next, we evaluate the use of senders with partial content, alone or supplementing full senders, and show the potential for speedups from parallel collaborative transfers. 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.

6.1 Simulation Parameters

All of our experiments focus on collaborative transfers of a 128MB file. We assume that the origin server divides this file into 95,870 input symbols of 1400 bytes each, and subsequently encodes this file into a large set of output symbols. We associate each output symbol with an identifier representing the set of input symbols used to produce it; our simulations used 64-bit identifiers. The irregular degree distribution used in the codes was generated using heuristics based on the discussion in Section 5.4 and described in [11]. This degree distribution had an average degree of 11 for the encoded 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 generated degree distributions in the same fashion with a maximum degree of 50. Rather than generate recoding degree distributions on the fly, we instead generated them off-line and parameterized by containment and the percentage of available symbols desired by the receiving peer, both in increments of 0.05. We note that using more sophisticated techniques for generating degree distributions and reducing decoding overhead such as those described in [17, 16] will improve our results accordingly. Min-wise summaries employed 180 permutations, yielding 180 entries of 64 bits each for a total of 1440 bytes per summary. Fine-grained reconciliation used Bloom filters with 6 hash functions and $8(1+\epsilon_d)$ bits per input symbol, for a total of 96 KB per filter. Overhead measurements presented in this section using the faster approximate reconciliation tree methods are visually indistinguishable from those using Bloom filters and are not included. Additional experiments comparing approximate reconciliation trees to Bloom filters will be detailed in a subsequent paper.

6.2 Collaboration Methods

We compare the following 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 illustrate the basic tradeoffs. The details of the scenarios are as follows.

Uninformed Collaboration The sending peer randomly picks an available symbol to send. This simple strategy is used by Swarmcast [30] and works best when working sets are uncorrelated.

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. This choice of distribution does not take into account correlation with other sending peers but will be at least as efficient as uninformed collaboration (arguably a special case) and frequently more so.

Reconciled Collaboration The sending peer uses either a Bloom filter or an approximate reconciliation tree from the receiving peer to filter out duplicate symbols and sends a random permutation of them without repetition. The Bloom filter and approximate reconciliation trees are made large enough to contain all of the output symbols at the end of the process since they will be updated incrementally as output symbols are recovered. Random permutations of the transmitted encoding symbols are used to minimize the likelihood that two distinct sending peers send identical encoding symbols to the receiving peer.

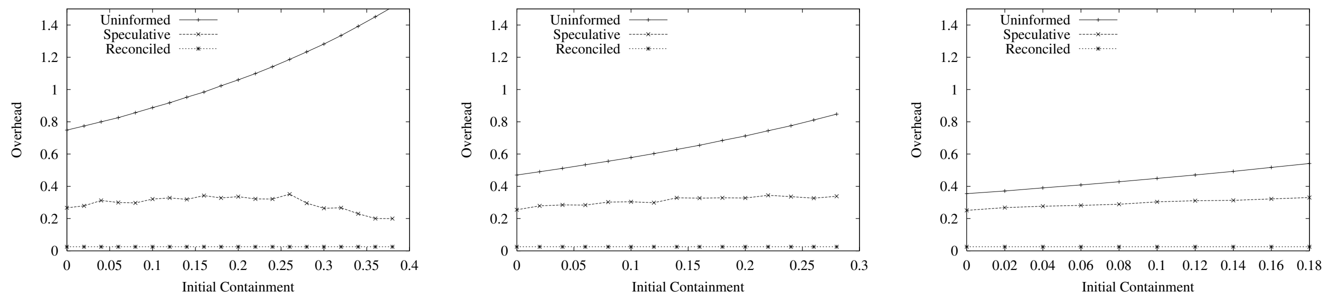
Techniques from speculative collaboration can be combined with the methods for reconciled collaboration to optimize performance over lossy channels or when transfers from peers with highly correlated working sets are employed in parallel.

6.3 Scenarios and Evaluation

In the scenarios we examine, we vary three components; 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 4 (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 output 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,



(a) Slack = 1.1 (b) Slack = 1.2 (c) Slack = 1.3

Figure 7: Overhead of peer-to-peer reconciliation.

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, 1.2, and 1.3 for comparison between compact scenarios with little available redundancy and looser scenarios. When varying slack has little effect on the results, only the results for a slack value of 1.1 are shown.

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 output symbols from the server. The output symbols known to the sending peers are determined by the slack of the scenario and the containment defined in Section 4; this will be discussed in detail for each particular scenario below.

To evaluate each technique, we measure the overall overhead of each strategy where an 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 duplicate or useless received encoding symbols or *recoding overhead* from 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.

6.3.1 Peer-to-Peer Reconciliation

The simplest scenario to consider is composed of two peers with partial content where one peer sends symbols to the other. This scenario is illustrated in Figure 6(a), and is designed to illustrate the feasibility of our approach even in the worst case when servers with a complete copy of the file are no longer available and reconciliation and recovery is barely possible.

For receiving peer A, sending peer B, with a file consisting of ℓ input symbols and slack s,

$$\ell s \quad = \quad |S_A| + |S_B| - |S_A \cap S_B|. \tag{1}$$

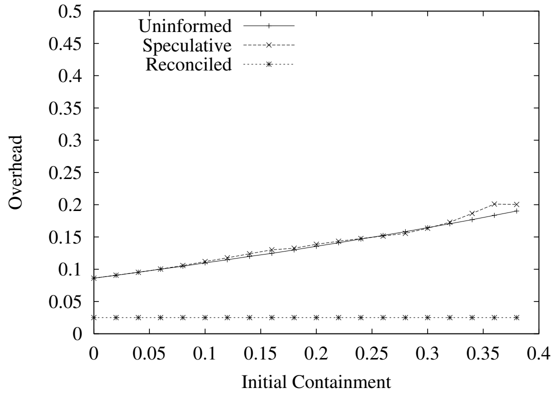


Figure 8: Overhead of peer-augmented downloads, slack = 1.1

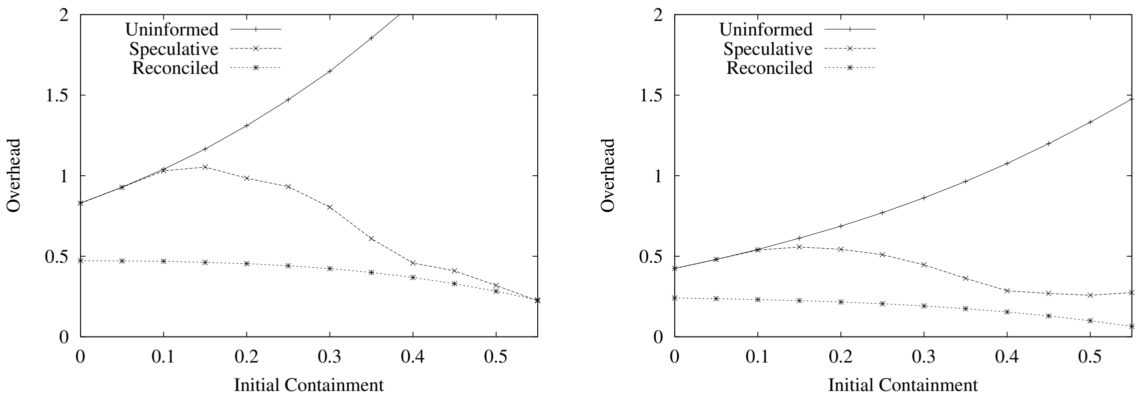
By the definition of containment, $c = \frac{|S_A \cap S_B|}{|S_B|}$,

$$|S_B| \quad = \quad \frac{\ell s - |S_A|}{1 - c}. \tag{2}$$

These two equations therefore uniquely determine $|S_A|$, $|S_B|$ and $|S_A \cap S_B|$ as a function of the slack and the containment. The $|S_A \cup S_B|$ symbols are then distributed as follows: $|S_A \cap S_B|$ symbols are distributed to both A and B, $|S_A| - |S_A \cap S_B|$ symbols are distributed to A, and the remainder are distributed to B.

Before continuing, we note that one additional constraint is needed to keep the scenarios realistic, namely, neither A nor B alone should be able to recover the file (otherwise, no transfer is necessary or B can generate fresh symbols). That is, $|S_A, S_B| \leq (1+\epsilon_d)\ell$, where ϵ_d is the decoding overhead. This gives an upper bound on feasible values of c for a given slack s, explaining the variation between values on the x-axes of our plots.

Figure 7 shows the results of our experiments for this scenario. In each experiment, uninformed collaboration performs poorly and degrades significantly as containment increases. This result is intuitive and can be precisely analyzed using analysis similar to that of the well known Coupon Collector's problem [14]. Essentially, the rate of useless symbols transmitted increases with the number of symbols shared between peers. The degree of sharing increases both as the initial containment increases and as the transfer progresses.



(a) Slack = 1.1

(b) Slack = 1.3

Figure 9: Overhead collaborating with multiple peers in parallel.

Speculative collaboration is more efficient than uninformed collaboration, but the overhead still increases slowly with containment. In comparison, the overhead of reconciled collaboration is virtually indistinguishable from plain encoded transfers from a server and does not increase with containment. The extra overhead of reconciled collaboration is purely from the cost of reconciliation (i.e. transmitting a Bloom filter or approximate reconciliation tree) so it is less than a percent when sending 8 bits for every symbol (1400 bytes).

6.3.2 Peer-Augmented Downloads

The next scenario we consider consists of a download from a server with complete content, supplemented by a perpendicular transfer from a peer as illustrated in Figure 6(b). In contrast to the previous scenario, this scenario demonstrates the utility of additional bandwidth in parallel with an ongoing download from a server. As in the case of peer-to-peer reconciliation, the distribution of symbols between peers at the beginning of the scenario is precisely determined by the slack and containment.

The results of this scenario are shown in Figure 8 and are similar regardless of the slack. The overhead of uninformed collaboration is considerably lower than in the scenarios of Figure 7, 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 – some improvements are possible with additional effort. In all cases, reconciled collaboration still has overhead just slightly higher than that of only receiving symbols directly from the server, but the transfer time is substantially reduced when the additional connection is employed.

For this scenario, it is natural to consider the speedup that is obtained by augmenting the download with an additional connection. Defining the speedup to be the ratio between the transfer time using a single sender with full content (and incurring no decoding overhead) and the transfer time we achieve, we have:

$$\text{speedup} = \frac{\text{number of senders}}{1 + \text{ overhead}},$$

since all connections are assumed to have equal bandwidth and are fully utilized. Therefore, a reconciled transfer with 0.025 overhead achieves a speedup of 1.95, while an uninformed transfer with 0.20 overhead achieves a more modest speedup of 1.67 over a vanilla download.

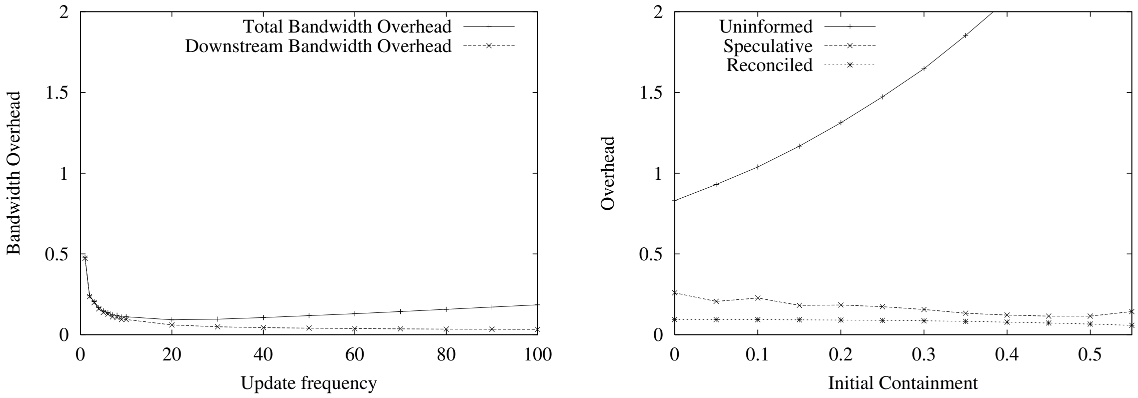
6.3.3 Collaborating with Multiple Peers in Parallel

Finally, we consider a peer collaborating concurrently with four peers, all with partial content, as illustrated in Figure 6(c). 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 employ the following allocation method. As before, the receiver initially has exactly 0.5ℓ symbols. One of these symbols is known to a sending peer with probability c. The remaining symbols are known 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 symbols not known to any sending peers is discarded and replaced. This results in each peer having an expected 0.5ℓ symbols at the beginning of the experiment

The results of this scenario are shown in Figure 9. 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. We again recall that the degree distribution was tuned to the according to the containment. In contrast to previous experiments, reconciled collaboration has much higher overhead than before. This arises from correlation across multiple peers. For example, 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.

Given the relatively poor performance of reconciled collaboration when there is sharing between sending peers, we now consider the



(a) Tradeoffs between bandwidth and update frequency.

(b) Overhead with updated summaries ($f = 10$).

Figure 10: Overhead of collaborating with multiple peers in parallel and updating periodically. Slack = 1.1.

effects of periodically updating the summaries, in contrast to the previous experiments, which performed fine-grained reconciliation only once, at the beginning of the scenario. We repeat the experiments for this scenario with the containment constrained to zero (the worst case for reconciled collaboration) and modulate the frequency of reconciliation. Figure 10(a) shows the results of this experiment. In this graph, the update frequency f means that an update is performed after receiving ℓ/f symbols, i.e. a frequency of 20 implies that updates are triggered after every 5% of the download progresses. The bottom curve reflects the extra bandwidth of traffic to the receiving peer. The top curve adds the bandwidth consumed by updates, thus accounting for the total amount of extra communication in *both* directions. For example, as f increases, the bandwidth spent on reconciliation updates becomes significant, and ultimately would dominate the bandwidth of the actual transfer. When optimizing total bandwidth consumption, we find that a reasonable reconciliation frequency is roughly $10 - 20$ depending on the slack of the scenario, meaning that there is an update after every $0.05\ell - 0.10\ell$ symbols that are transferred.

Figure 10(b) shows the results of using these updates in the scenarios of Figure 9, i.e. speculative collaboration updates the min-wise summary and reconciled collaboration updates the Bloom filters. An update frequency of 10 is used and both speculative and reconciled collaboration show dramatic improvement.

7 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.

Acknowledgements

We would like to thank Ari Trachtenberg and the anonymous SIGCOMM '02 reviewers for the helpful feedback they provided on earlier versions of this paper.

References

[1] Altavista. www.altavista.com.

[2] ANDERSEN, D., BALAKRISHNAN, H., KAASHOEK, F., AND MORRIS, R. Resilient overlay networks. In *Proc. of ACM Symposium on Operating Systems Principles* (Banff, Canada, October 2001).

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

[4] BOHMAN, T., COOPER, C., AND FRIEZE, A. Min-wise independent linear permutations. *Electronic Journal of Combinatorics 7, R26* (2000).

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

[6] 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.

[7] 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.

[8] BYERS, J. W., LUBY, M., MITZENMACHER, M., AND REGE, A. A digital fountain approach to reliable distribution of bulk data. In *Proc. of ACM SIGCOMM* (Vancouver, September 1998), pp. 56–67. To appear in IEEE Journal on Selected Areas in Communications.

[9] CHAWATHE, Y. *Scattercast: An Architecture for Internet Broadcast Distribution as an Infrastructure Service*. PhD thesis, University of California, Berkeley, December 2000.

[10] CHU, Y.-H., RAO, S., AND ZHANG, H. A case for end system multicast. In *ACM SIGMETRICS* (Santa Clara, CA, June 2000).

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

[12] FAN, L., CAO, P., ALMEIDA, J., AND BRODER, A. Summary cache: A scalable wide-area cache sharing protocol. *IEEE/ACM Trans. on Networking 8(3)* (2000), 281–293. A preliminary version appeared in Proc. of SIGCOMM '98.

[13] 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).

[14] KLAMKIN, M., AND NEWMAN, D. Extensions of the birthday surprise. *Journal of Combinatorial Theory 3* (1967), 279–282.

[15] LABOVITZ, C., MALAN, G., AND JAHANIAN, F. Internet routing instability. In *Proc. of ACM SIGCOMM* (September 1997).

[16] LUBY, M. Information Additive Code Generator and Decoder for Communication Systems. U.S. Patent No. 6,307,487, October 23, 2001.

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

[18] MACWILLIAMS, F. J., AND SLOANE, N. *The Theory of Error-Correcting Codes*. North Holland, Amsterdam, 1977.

[19] MAHANTI, A., EAGER, D. L., VERNON, M. K., AND SUNDARAM-STUKEL, D. Scalable on-demand media streaming with packet loss recovery. In *Proc. of ACM SIGCOMM* (August 2001), pp. 97–108.

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

[21] MINSKY, Y., AND TRACHTENBERG, A. Practical set reconciliation. Tech. Rep. BU ECE 2002-01, Boston University, 2002.

[22] 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).

[23] MITZENMACHER, M. Compressed bloom filters. In *Proc. of the 20th Annual ACM Symposium on Principles of Distributed Computing* (2001), pp. 144–150. To appear in IEEE/ACM Trans. on Networking.

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

[25] RATNASAMY, S., FRANCIS, P., HANDLEY, M., KARP, R., AND SHENKER, S. A scalable content-addressable network. In *Proc. of ACM SIGCOMM* (San Diego, CA, August 2001).

[26] RODRIGUEZ, P., AND BIERSACK, E. W. Dynamic parallel-access to replicated content in the Internet. *IEEE/ACM Transactions on Networking 10(4)* (August 2002). A preliminary version appeared in Proc. of IEEE INFOCOM '00.

[27] ROWSTRON, A., AND DRUSCHEL, P. Storage management and caching in past, a large-scale, persistent peer-to-peer storage utility. In *Proc. of ACM Symposium on Operating Systems Principles* (Banff, Canada, October 2001).

[28] 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).

[29] STOICA, I., MORRIS, R., KARGER, D., KAASHOEK, F., AND BALAKRISHNAN, H. Chord: A scalable peer-to-peer lookup service for internet applications. In *Proc. of ACM SIGCOMM* (San Diego, CA, August 2001).

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

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

[32] ZHUANG, S., ZHAO, B., JOSEPH, A., KATZ, R., AND KUBIATOWICZ, J. Bayeux: An architecture for scalable and fault-tolerant wide area data dissemination. In *Proc. of NOSSDAV '01* (Port Jefferson, NY, June 2001).