2002

Informed Content Delivery Across Adaptive Overlay Networks

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, multipoint transfers across richly connected overlay networks, focusing on the question of *what* to put in each transmitted packet. We first make the case for transmitting encoded content in this scenario, arguing for the digital fountain approach which enables end-hosts to efficiently restitute the original content of size n from a subset of *any* n symbols from a large universe of encoded symbols. Such an approach affords reliability and a substantial degree of application-level flexibility, as it seamlessly tolerates packet loss, connection migration, and parallel transfers. However, since the sets of symbols acquired by peers are likely to overlap substantially, care must be taken to enable them to collaborate effectively. We provide 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 messaging 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.

1 Introduction

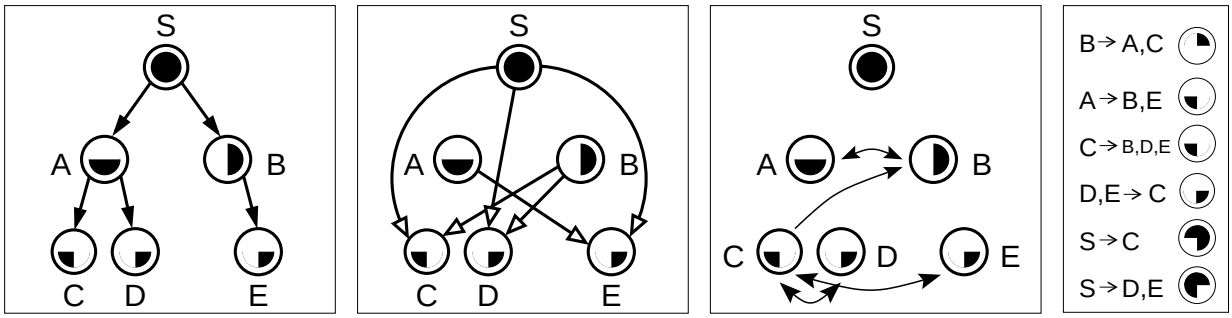
We motivate our work with a representative example. Consider the problem of distributing a large new file across a content delivery network of several thousand geographically distributed machines. Transferring the file with individual point-to-point connections incurs two performance limitations. First, 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. The first problem of excessive bandwidth consumption can be solved by a reliable multicast-based approach. With multicast, a network element may forward a single inbound packet payload across multiple outbound links, thus in the absence of packet loss, a single copy of each packet payload transmitted by the server traverses each link in the multicast tree to all members of the multicast group. Providing reliability is another challenge, but one elegant solution is the digital fountain approach [7], whereby the content is first stretched into a loss-resilient encoding [17, 22, 16], then transmitted to clients. This approach tolerates asynchronous arrivals, heterogeneous client transfer rates (if layered multicast is also employed) and packet loss.

Although multicast-based dissemination offers near-optimal scalability in bandwidth and server load, IP multicast suffers from limited deployment. This lack of deployment has led to the development of *end-system* approaches [9, 13, 8], along with a wide variety of related schemes relevant to peer-to-peer content delivery architectures [23, 10, 26, 29, 12, 20, 28]. Many of these architectures overcome the deployment hurdle faced by IP multicast by requiring no changes to routers nor additional router functionality. Instead, they construct *overlay* topologies of unicast connections, typically connecting end-systems, and map these topologies onto the underlying physical network.

End-system multicast differs from IP multicast in a number of fundamental aspects. First, overlay-based approaches may redundantly map multiple virtual paths onto the same network path. Second, unlike IP multicast trees, overlays may flexibly adapt to changing network conditions. For example, overlays may reroute around congested or unstable areas of the Internet [2, 27]. And third, end-systems are now explicitly required to cooperate. This latter 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.

Traditional service models which employ tree topologies are bandwidth-limited, as the transfer rate to a client will only be as fast as the throughput of the bottleneck link on the path from the server. Unlike other models, overlay networks are capable of overcoming this limitation. We argue that, in systems with ample bandwidth, the performance of overlay networks can substantially benefit from additional connections between end-systems. Such improvement is possible due to intelligent collaboration in making effective use of the extra available bandwidth. 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.

*This work was supported in part by NSF CAREER awards CCR-9983832 and ANIR-0093296, NSF grants ANIR-9986397, CCR-0118701, CCR-0121154, and an Alfred P. Sloan Research Fellowship.



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

Figure 1: **Possibilities for file delivery.** (Shaded content within a topology node represents the working sets of nodes. Connections of (b) supplement (a), connections of (c) supplement (a)+(b). S contains full content. A, B store a different 50% of the total content. C, D, E each hold 25% of total content. The working sets of C and D are disjoint.)

Our argument is illustrated by the content delivery scenario of Figure 1(a), where the root of the tree is the source and all other nodes in the tree represent end-systems attempting to download a large file. Each node has a *working set* of packets, the subset of packets it has received. In Figure 1(a), even if the overlay management of the end-system multicast 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* [6], resulting from connection to multiple servers concurrently (Figure 1(b)). More generally and more importantly, drastic performance benefits may be obtained by taking advantage of "perpendicular" bandwidth among nodes whose working sets are complementary, as pictured on Figure 1(c). In this example, we assume the content is not encoded (although better performance is possible when it is), and the portions of content which can be beneficially exchanged between end-systems is shown on the legend of 1(d).

The tree and DAG topologies of Figures (1(a),1(b)) impede the full flow of content to downstream receivers, as the rate of flow monotonically decreases along each path away from the source. In contrast, the opportunistic connections of the graph of Figure 1(c) allow for much higher transfer rates, but simultaneously demand a much more careful level of orchestration between end-systems to achieve those rates. In particular, any given end-system in a peer-to-peer relationship must be able to determine *which* subsets of their blocks their peer lacks, and subsequently make an *informed transfer* of those subsets, made possible by *reconciliation* of their two working sets.

When working sets are limited to small groups of contiguous blocks of sequentially indexed packets, reconciliation is trivial, since a block can be succinctly represented by the smallest index and an offset. However, restricting working sets to such simple patterns greatly limits flexibility to the frequent changes which arise in adaptive overlay networks, as we will argue in Section 2. We also argue that a more attractive and flexible alternative is to use encoded content, and we provide the rationale for encoded content and elaborate the numerous benefits of using this approach in Section 2. However, the main drawback of encoded content is that reconciliation becomes a challenging problem. To address this point, in Sections 3, 4, and 5, we provide a set of tools for estimating, summarizing, and approximately reconciling of working sets of connected clients, all of which keep messaging 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 in Adaptive Overlays

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

In 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.
- **Scalability:** The service must scale to large receiver populations and large content.
- **Transience:** Routers, links, and end-systems may fail, or their performance may fluctuate.

Overlay networks should tolerate asynchrony and heterogeneity and should adapt to transient behavior, all in a scalable manner. For

example, a robust adaptive overlay network should have the ability to 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, 27]. Such reactive behavior of the virtual topology may frequently force the nodes to reconnect to better-suited peers. But of course this behavior exacerbates all of the other fundamental problems enumerated above.

Another significant consequence of the fluidity of the environment is that content is likely to be disseminated non-uniformly across peers. Significant discrepancies between working sets are likely to arise due to uncorrelated losses, bandwidth differences, asynchronous joins and topology reconfigurations. For example, receivers with higher transfer rates and receivers who arrive earliest will simply have more content than their peers, while receivers with uncorrelated losses will feature gaps in different portions of their working sets. 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.

As we have argued in the introduction, we also want to take advantage of a significant *opportunity* presented by overlay networks: the ability to download content from multiple end-systems in parallel. This raises the further challenge of how to deliver "useful" content across multiple end-system paths. A similar opportunity arises from the adaptive nature of robust overlay networks: can we make beneficial use of ephemeral connections which may be short-lived, may be preempted, or whose performance may fluctuate.

2.2 Limitations of Stateful Solutions

We argue that it is possible to address all of the problems and concerns described in the preceding subsection, but it cannot be done trivially. In particular, solutions to these goals cannot be scalably achieved with techniques that require state to be stored at connection endpoints. For example, while handling issues of connection migration, heterogeneity, and asynchrony is 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 in the face of transience, since this state must be maintained in the face of reconfiguration and reconnection. We will later describe how the scalability and performance problems caused by per-connection state can be avoided using encoded content, following several papers describing the use of forward error correction for multicast [24, 21, 7].

Parallel downloading using stateful approaches is also problematic, as discussed in [6]. The natural approach is to divide the range of the missing packets into disjoint sets in order to download different ranges from different sources. In the face of 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 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 [25]), encoding allows much simpler and more effective parallel downloading.

The settings of overlay networks introduce an additional problem: in order for useful content to be obtained from multiple sources, we actually *prefer* uneven distribution of content across participating end-systems. As noted earlier, discrepancies in working sets will naturally arise due to factors such as uncorrelated losses, bandwidth differences, asynchronous joins, and topology reconfigurations. If each end-system aims to obtain a consecutive prefix of unencoded packets, however, the ability to transfer useful content between end-systems may be limited, 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 running over an unreliable transport protocol. The digital fountain approach [7] 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 code* [16, 17, 22] 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 3-5% percent more than the number of symbols in the original file, 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.

- **Stateless Encoding:** Senders with a copy of a file may continuously produce a streamed encoding of its content.
- **Time-Invariance:** Encoding symbols are produced by a digital fountain in a memoryless manner, thus the content of a given stream is time-invariant.
- **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, thus parallel downloads from multiple sources in possession of full 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 for by an encoding method which frees the receiver from receiving all of

a set of distinct symbols enables fully *stateless* connection migrations, in which no state need be transferred among hosts and no dangling retransmissions need be resolved. It also allows the nodes of the overlay topology to connect to as many senders as necessary and obtain distinct pieces of encoding from each, provided these senders are in possession of the entire file.

Given the advantages of using encoded content, we turn to the potential disadvantage, aside from the small overhead associated with encoding and decoding operations. In a setting where encoded content comes from a *large, unordered* universe instead of a sequentially ordered stream, 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. But it does not solve this problem when an end-system can only forward received encoded packets, since the receiving end-system may already have obtained the same encoded packet. A significant fraction of transfers may fall into the latter setting, especially when delivering large files. To avoid redundant transmissions in such scenarios, nodes require mechanisms for estimating and reconciling the differences in their working sets and subsequently performing *informed transfers* of missing content. Moreover, in the environment featuring frequent reconnections of nodes with partial content, driven by overlay management, efficiency of the reconciliation mechanism is paramount to the overall performance of the content delivery system.

3 Reconciliation and Informed Delivery

The preceding sections have defined the settings for informed collaboration: an adaptive content delivery architecture designed for transmission of large files, using erasure codes. We abstract our solutions from the issues of optimization of the overlay, as well as distributed naming and indexing.

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 messaging complexity. In the subsequent sections, we propose the following approaches:

Coarse-grained reconciliation using working set sketches, by random sampling or by minwise permutations. 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 working set symbols to re-encode and transfer. This process can be fine-tuned using results of coarse-grained reconciliation.

Fine-grained reconciliation using compact, searchable working set summaries, either Bloom filters or approximate reconciliation trees. Fine-grained approaches are more resource-intensive. They allow a peer to determine the symbols in the working set of another peer, with some degree of certainty.

Reconciled transfers entail a sender filtering the transmissions of packets deemed redundant by the fine-grained reconciliation summaries.

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 10KB in size. Finally, re-encoded content makes additional resource demands on the end-system as it requires a node to blend received symbols together (albeit using fast XOR operations) to produce new symbols for transmission. Re-encoding content facilitates optimized 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 the overlap of the working sets of pairs of nodes prior to establishing connections. Knowledge of the degree of working set correlation allows a receiver to determine the extent to which a prospective peer offers useful content. Additionally, as we show in Section 5.4, the magnitude of the working set resemblance is useful for the recoding strategy we develop. In our context, it is essential that the data be conveyed between the peers to compute the resemblance as efficiently as possible. Our methods are designed to give accurate answers when a single 1KB packet of data is transferred between nodes. We emphasize that there are different tradeoffs involved in each of these approaches, which 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 A_F and B_F containing symbols from an encoding of the file F. The quantity $\frac{|A_F \cap B_F|}{|B_F|}$ represents the fraction of elements B has that can be useful to A. If this quantity is close to zero, the overlap is small, and B rates to be a useful source of information. In this section, we suppose that the each element of the working sets of peers is identified by an integer key; when we refer to sending an element, we mean sending this integer key. We will think of these keys as unique, although they may not be; for example, if the keys are determined by a hash function, it may be possible (with small probability) for two elements to have the same key. This will introduce small errors in estimating the overlap; for most applications, since we care only in the approximate magnitude of the overlap, these small errors will not matter. If element keys are 64 bits long, then a 1KB packet can hold roughly 128 keys, which enables sufficiently accurate estimates for all techniques we

describe. Finally, note that we may assume that the integer keys are 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. Optionally, we may also send the size of the working set, although this is not essential for all of the techniques we describe below. Suppose A sends B a random sample A_k from A_F. The probability that each element in A_k is also in B_F is $\frac{|A_F \cap B_F|}{|B_F|}$, and hence $\frac{|A_k \cap B_F|}{k}$ is an unbiased estimate of this quantity. Random sampling suffers the drawback that B must search for each element of A_k in its own list B_F. Although such searches can be implemented quickly using standard data structures (interpolation search will take $O(\log\log|B_F|)$ average time per element), this requires some overhead in keeping the data structure up to date. As another consideration, the computation may create a bottleneck by delaying the return of the overlap measure from B to A. Another concern about random sampling is that it does not easily allow a peer to check overlap among multiple peers. For example, if peer A is attempting to establish connections with peers B and C, it might be helpful to know the overlap between the working sets of B and C. Random sampling does not allow this.

The next two alternatives, suggested by Broder [4], make use of more clever sampling techniques. These techniques were designed previously to determine the similarity of documents in search engines [1], and have database applications as well. The first approach is to sample by taking elements whose keys are 0 modulo k for an appropriately chosen k, yielding samples A_k and B_k. (Here we specifically assume that the keys are random.) The aim is to make these samples constant size. In this case $\frac{|A_k \cap B_k|}{|B_k|}$ is an unbiased estimate of $\frac{|A_F \cap B_F|}{|B_F|}$. Here, all computation can be done directly on the small samples, instead of on the working sets. This technique suffers from the problem that the samples are of variable size, which complicates matters in practice, since packets have a maximum size. There are also some difficulties when the working sets are dramatically different in size, although there are ways of handling such problems.

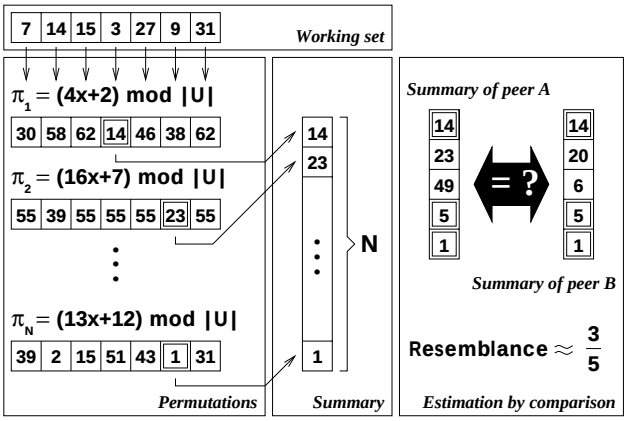


Figure 2: Example of minwise summarization and estimation of resemblance (Key universe size is 64, example permutation functions shown).

The final approach, which avoids the drawbacks of the first two approaches and which we prefer, calculates working set resemblance based on sketches, following [4, 5]. 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 A_F and B_F containing symbols of the file F, we have $x = \min \pi_j(A_F) = \min \pi_j(B_F)$ if and only if $\pi_j^{-1}(x) \in A_F \cap B_F$. That is, the minimum element after permuting the two sets A_F and B_F matches only when the inverse of that element lies in both sets. In this case, we also have $x = \min \pi_j(A_F \cup B_F)$. If π_j is a random permutation, then each element in $A_F \cup B_F$ is equally likely to become the minimum element of $\pi_j(A_F \cup B_F)$. Hence we conclude that $\min \pi_j(A_F) = \min \pi_j(B_F)$ with probability $r = \frac{|A_F \cap B_F|}{|A_F \cup B_F|}$. The quantity r is a number in the range 0 to 1 that represents the resemblance between the two sets. Note that r is different than the quantity $\frac{|A_F \cap B_F|}{|B_F|}$ determined by the other techniques; however, given $|A_F|$ and $|B_F|$, an estimate for one can be used to calculate an estimate for the other, by using the inclusion-exclusion formula. To estimate the resemblance, the peer A computes $\min \pi_j(A_F)$ for some fixed number of permutations π_j (as shown on Figure 2), and similarly for B and B_F. The peers must agree on these permutations in advance; we assume they are fixed universally off-line.

For B to estimate $\frac{|A_F \cap B_F|}{|A_F \cup B_F|}$, 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 on Figure 2. The result is an accurate 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, without dramatically affecting overall performance [5].

The minwise 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 A_F and B_F is easily found by taking the coordinate-wise minimum of $v(A)$ and $v(B)$. Thus to estimate the overlap of a third peer's working set C_F with the combined working set $A_F \cup B_F$ can be done with $v(A), v(B)$, and $v(C)$.

All of our approaches can be incrementally updated upon acquisition of new content, with constant overhead per receipt of each new element and hence these methods estimating overlap can function even as new data arrives at the peers.

Methods for similarity comparison described in this section, with high degree of certainty, expose the situations when content in the working sets of peers is identical. Such methods are suitable for simple admission control, allowing receivers to immediately reject candidate senders whose content is identical to their own. The receivers will also be able to distribute the load among the senders whose content is identical, as shown by the comparison of the summaries submitted by all the sender candidates. Equipped with similarity estimation, overlay management may explicitly avoid connecting nodes with identical content.

5 Reconciling Differences

As shown in the previous section, a single packet can allow peers to estimate the overlap in their working sets. If the difference is sufficient to allow useful exchange of data, the next step is for one peer to determine what data would be useful for the other. Again, the goal is to make this step as simple and efficient as possible. The next set of methods we provide generally require transmission of a handful of packets. Here there are also a number of performance considerations that we develop below.

The problem above is simply a set difference problem. Specifically, suppose peer A has a set S_A and peer B has a 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 $S_A - S_B$ as possible.

The set difference problem has been widely studied in communication complexity. The focus, however, has generally been on determining the exact difference $S_B - S_A$. In our setting, because the data is encoded, a peer does not necessarily need to get all of the symbols in this difference. For example, if the two peers both have 3/4 of the symbols necessary to reconstruct the file, and there is no overlap between them, then only a small amount of the difference needs to be sent. More generally, in this setting we do not need exact reconciliation of the set difference; very weak 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 and easily determine approximate differences using Bloom filters [3]. We also introduce a new data structure, which we call an approximate reconciliation tree, that uses Bloom filters. 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) in bits.
- Computation time to determine the approximate set difference.
- Inaccuracy of the approximation, measured by the number of elements in the difference, but not identified as such.

Known approaches that provide perfect accuracy, and that we describe briefly next, are prohibitive in either computation time or transmission size. Bloom filters and our approximate reconciliation tree trade off accuracy against transmission size and computation time.

5.1 Exact Approaches

To compute differences exactly, peer A can obviously send the entire 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 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 n 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 [19]. 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. The preprocessing time, however, involves $\Theta(d|S_A|)$ field operations in a field of size u (or h, if hashing is used), and the work to determine the discrepancy is $\Theta(d^3)$. This approach therefore is prohibitive except when d is known and known to be small (like 100 or less).

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 $|S_B - S_A|$. This allows us to do significantly better than exact approaches in practice, especially when $|S_B - S_A|$ is large. We describe how to use Bloom filters in this case.

We first review the Bloom filter data structure [3]. More details can be found in [11]. 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: $f = (1 - \mathrm{e}^{-kn/m})^k$.

In our setting, Bloom filters provide a simple approximate solution. Peer A sends a Bloom filter B_A of S_A; peer B would then check for each element of S_B in B_A. When a false positive occurs, peer B would mistakenly think that peer A has a symbol that it does not have, and peer B ends up not sending 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 problem, especially for encoded content where the universe of symbols greatly exceeds the quantity necessary for reconstruction.

The number of bits per element can be kept small while still achieving a suitable false positive ratio. For example, using just four bits per element and three hash functions yields a false positive probability of 14.7%; using eight bits per element and five hash functions yields a false positive probability of 2.2%. Using four bits per element, we can create filters for 10,000 packets using just 40,000 bits, which can fit into five 1 KB packets. More generally, $O(|S_A|)$ preprocessing is required to set up the Bloom filter, and $O(|S_B|)$ work to find the 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, if $|S_A|$ and $|S_B|$ are larger than tens of thousands, then peer A can create a Bloom filter only for elements of S 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

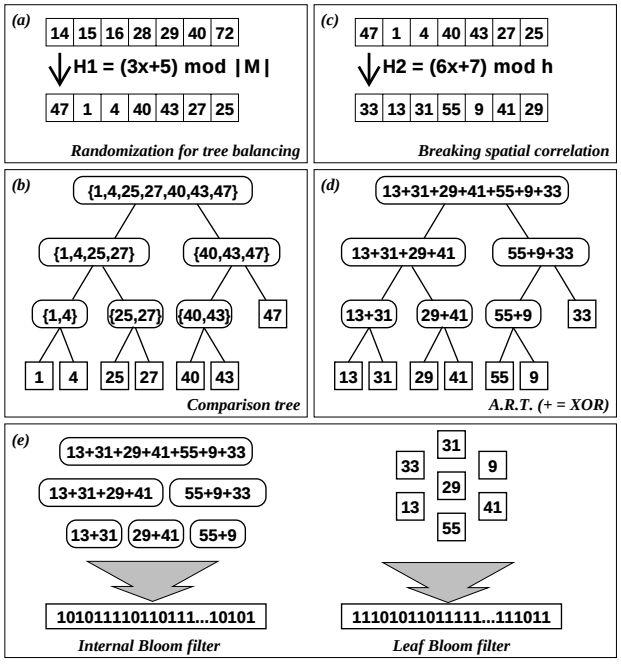


Figure 3: Example of creation and Bloom filtering of an approximate reconciliation tree. (M is $O(\text{poly}\ |S_A|)$; in this case, $M = |S_A|^2 = 49$ h is 64, and example permutation functions are as shown.

Bloom filters are the preferred data structures when the working sets of the two peers have small overlap. However, our overlay approach can be useful even when the overlap 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 [18]. Our data structure has several useful properties and other applications beyond that which we describe here, that will be detailed in a subsequent paper. We limit ourselves here to an introductory description focused on our applications.

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 set S_A. The children correspond the 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 such a tree. 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 $\log u$.

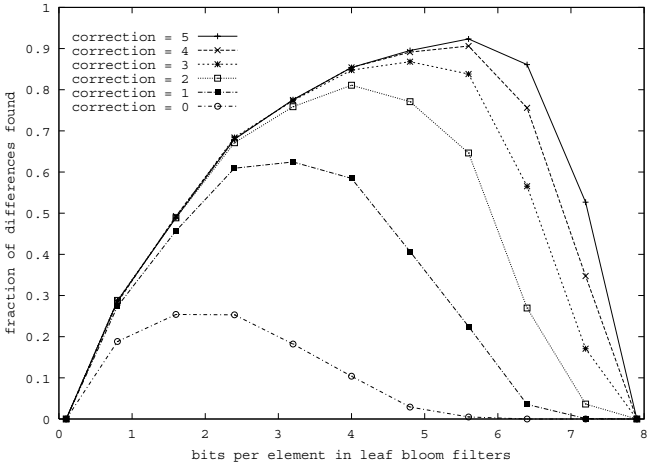
The above tree has $\Theta(u)$ nodes and depth $\Theta(\log u)$, which is unsuitable when the universe is large. However, almost all the nodes in the tree correspond to the same sets. In fact there are only $O(|S_A|)$ non-trivial nodes. The tree can be collapsed by removing trivial edges between nodes that correspond to the same set, leaving only $O(|S_A|)$ nodes. Unfortunately, the depth may still be $O(|S_A|)$. 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 $O(\log |S_A|)$ 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 $O(n)$ 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. Each leaf element is hashed again into a universe $U' = [1, h)$ to avoid spatial correlation, particularly in the higher order bits. 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). 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 do to the hashing. As with Bloom filters, false positives may cause peer B to miss some nodes in the difference $|S_B - S_A|$.

The advantage of the tree over a Bloom filter is that it may allow 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 the $O(|S_B|)$ for the Bloom filter. To avoid some bulkness 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 a reasonable accuracy.

A false positive from the Bloom filter cuts off the search for elements in the difference $S_B - S_A$ along tree path. If the false positive rate is high, the approximate reconciliation tree may never follow a full path down to the leaf. To handle this, we separate the leaf hashes from the internal hashes and to use separate Bloom filters, thus allowing the relative accuracies to be controlled. We ameliorate this weakness by not cutting off a search at the first match between nodes. Instead, we add a correction level indicating the number of

consecutive matches allowed without pruning the search. A correction level of 0 stops the search stops at the first match found while a correction level of 1 allows one match at an internal node but stops if a child of that node also matches. Figure 4(a) demonstrates both the tradeoff involved when shifting 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. Table 4(b) shows the accuracy for various numbers of bits per element and levels of correction using the optimal distribution of bits between leaves and interior nodes.



(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	$8n$	98%	$O(n)$
A.R.T. (correction=5)	$8n$	92%	$O(d \log n)$

(c) High level structure comparison at 8 bits per element

Figure 4: Approximate Reconciliation Statistics

5.4 Recoded Content

When two end-systems decide to form a peering relationship, they initially have very little information about one another's working sets. But often, peers who could collaborate productively have working sets with a high degree of correlation; for example, spatially proximate nodes in a layered multicast session may have received many common symbols over layers to which they have both subscribed. In this case, majority of individual symbols in possession by the peers is unlikely to be useful to others, making sharing ineffective. This situation is easily diagnosed with summaries whose use we advocated in Section 4. Unfortunately, using knowledge of correlation to improve the performance of naive delivery strategies is more difficult than its diagnosis. For example, the min-wise summaries only measure the magnitude of correlation but say almost nothing about the symbols in common. Approximate reconciliation trees and the other techniques mentioned in Section 5 provide more knowledge, but not all clients will have the processing capability to perform fine-grained reconciliation. We begin with additional details of erasure code constructions in Section 5.4.1, then describe effective modifications of these codes to provide 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 digital fountains. 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 source blocks. An *encoder* produces a potentially unbounded sequence of symbols, or encoding packets, $y_1, y_2, \ldots$ from the set of source blocks. With parity-check codes, each symbol is simply the bitwise XOR of a specific subset of the source blocks. To optimize decoding, the distribution of the size of the subsets chosen for encoding is irregular; a heavy-tailed distribution was proven to be a good choice in [16]. We say that an encoding is a *memoryless* encoding if the random subset of source blocks used to produce each encoding symbol is generated identically and independently from the same distribution. A *decoder* attempts to recover the content, i.e. the entire set of data packets symbols from a subset of the encoding symbols. For a given symbol, we refer to the number of source blocks 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 source blocks 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 [16]. While heavy-tailed distributions introduce some symbols of large degree, it is important to note that encoding and decoding times are a function of the *average* degree, not the maximum. When the average degree is constant, we say the code is sparse.

Finally, as mentioned earlier, sparse parity check codes typically require recovery of a small number (3% - 5%) of symbols beyond ℓ, the minimum needed for decoding. The *decoding overhead* of a code is defined to be $1 + c$ if $(1 + c)\ell$ encoding symbols are needed on average to recover the original content.

5.4.2 Recoding Methods

Suppose we have two peers, and peer A wishes to communicate useful (non-redundant) information to peer B. Assume that peer A has a set of symbols Y_A, of which a fraction q belong in the set Y_B of peer B. If peer A simply transmits a random symbol from Y_A to Y_B, that symbol will be redundant with probability q.

To mitigate problems of redundant transmissions, we generate *recoded* symbols. 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 symbols blended to create it. An encoded symbol must specify the *source blocks* from which it was generated; a recoded symbol must enumerate the *encoded symbols* from which it was produced. Although we do not provide a detailed discussion here, these lists can be stored concisely in packet headers. As with normal sparse parity check codes, irregular degree distributions work well, though we advocate use of a fixed degree limit primarily to keep the listing of identifiers short. This allows encoding and decoding to be performed in a fashion analagous to the substitution rule defined on normal sparse parity check codes. 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.

A wide range of analysis can be applied in designing a *provably* good degree distribution [16]. Our experience has been that heuristic approaches to constructing degree distributions also perform well in practice. Our heuristics employ some of the key ideas used in developing provably good distributions, especially the use of irregular degree sequences, which ensure that some high degree symbols are present, and tend to avoid low degree symbols, which may provide short-term benefit, but which are often useless.

To get a feel for the probabilities involved, we consider a representative calculation of interest, that of how to generate a recoded symbol which is *immediately* useful with maximum probability. Assume peer B is generating recoded symbols from file F for peer A and by virtue of a transmitted sketch, knows the value $c = \frac{|A_F \bigcap B_F|}{|B_F|}$. The probability that a recoded symbol of degree d immediately gives a new regular symbol is $\frac{\binom{(1-c)n}{d-1} cn}{\binom{n}{d}}$, where $n = |B_F|$. This is maximized for $d = \left\lceil \frac{|B_F|(1-c)+1}{|B_F|c} \right\rceil$. (Note that as recoded symbols are received, correlation naturally increases and the target degree increases accordingly.) While using this formula for d maximizes the probability of immediate benefit, choosing the locally optimal d is not necessarily globally optimal. As described earlier, 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 degrees generated, and generate degrees between this value and the maximum allowable degree, inclusively. As with regular encoded symbols, recoded symbols which are not immediately useful are often *eventually* useful — with the aid of recoded (or encoded) symbols which arrive later, they can subsequently be reduced to regular encoded symbols. By increasing the degree at the cost of immediate benefit, the probability of completely redundant symbols is greatly reduced.

6 Experimental Results

Our experiments focus on showing the overhead and potential speedups of using our methods in peer-to-peer transfers as well as in the settings of parallel downloads. We first show the feasibility of using a sender with partial content, by demonstrating the reconstruction overhead in receiving symbols from such a sender. Next, we evaluate the use of partial senders, alone or supplementing full senders, and show that parallel download speedups close to those of full senders are achievable.

6.1 Coding Parameters

The sparse parity check codes used were irregular, memoryless and tuned for up to 500K symbols using heuristics based on the discussion in Section 5.4.2. The degree distribution for recoding was created similarly with a degree limit of 50. A 32MB test file was divided into 23,968 source blocks of 1400 bytes, and subsequently encoded into output symbols, where the associated degree sequence representations of these symbols were 64 bits. The degree distribution used had an average degree of 11 for the encoded symbols and average decoding overhead of 6.8%. The experiments used the simplifying assumption of a constant decoding overhead of 7%. We note that using more sophisticated techniques for generating distributions such as those of [16] will slightly improve all of our results.

The final coding parameter is the domain of symbols over which recoding is performed. In the more oblivious cases, i.e. where no summary information or only minwise summaries were provided, recoding is performed choosing from all available symbols. On the other hand, when using Bloom filters or approximate set reconciliation trees, a partial sender can find symbols of guaranteed utility (ignoring parallel downloads) to the receiver. Thus recoding is not generally necessary in this situation. However, when updates are infrequent, recoding does afford some advantages when used in conjunction with summaries, in this case we restrict the recoding domain to an appropriate small size. In fact, in our experiments, we *never* send updates to our Bloom filter — doing so would of course provide a commensurate improvement in our transfer time.

In our experiments, the receiver may specify the number of symbols desired from each sender with appropriate allowances for decoding overhead. This number does not change during the simulation. In a full system, these estimates as well as other messages, including sketches, summaries or other control information, would be passed periodically.

6.2 Comparing Strategies

We compare the following strategies:

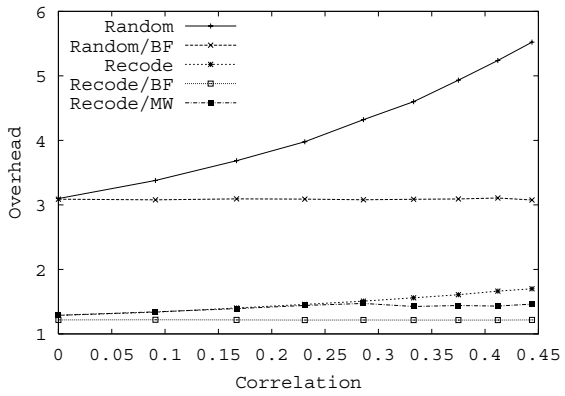
Random Selection (Random): The transmitting node randomly picks an available symbol to send. This simple strategy is used by Swarmcast [28] and works well when working sets are relatively uncorrelated.

Random Selection with Bloom filters (Random/BF): The sender selects symbols at random and sends those which are not elements of the Bloom filter provided by the receiver.

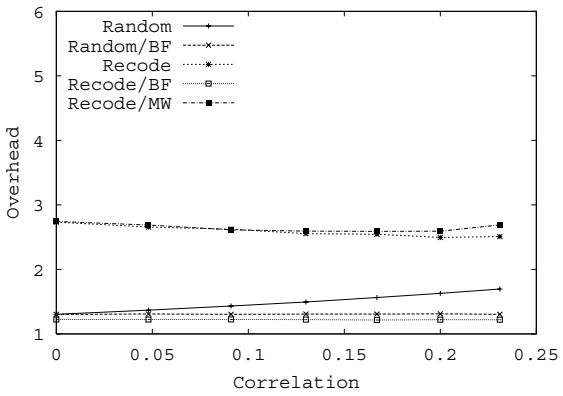
Recoding (Recode): Recoded symbols are generated as described in Section 5.4.2. Recoded symbols are generated from the entire working set of the partial sender.

Recoding with Bloom filters (Recode/BF): The previous strategy is augmented so that the recoded symbols are generated only from the symbols which are not in the Bloom filter from the receiver.

Recoding with Minwise Summary (Recode/MW): Recoded symbols are generated as described using a different degree distribution. Let $c = \frac{A_F \bigcap B_F}{B_F}$ as estimated using the

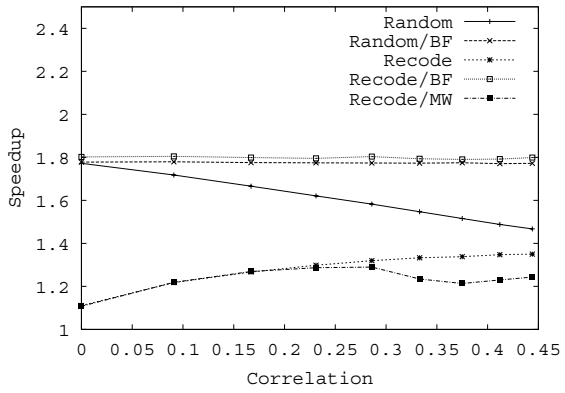


(a) Compact Scenarios, $1.1n$ distinct symbols in system

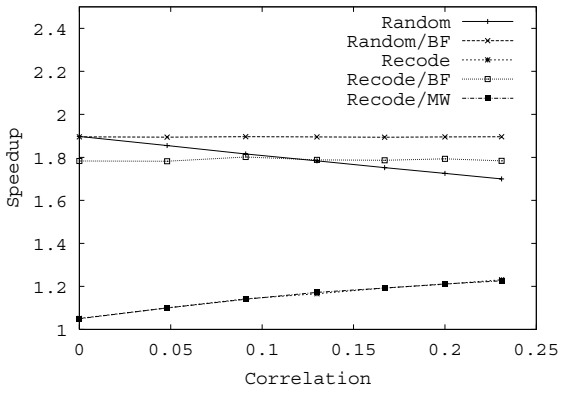


(b) Stretched Scenarios, $1.5n$ distinct symbols in system

Figure 5: Overhead of peer-to-peer transfers following various methods for reconciliation.



(a) Compact Scenarios, $1.1n$ distinct symbols in system



(b) Stretched Scenarios, $1.5n$ distinct symbols in system

Figure 6: Speedup in the rate of transfer of a receiver downloading from a full sender and a partial sender concurrently.

minwise summary where A is the receiver and B is the partial sender. If the regular recoding algorithm randomly generates a degree d symbol, generate a recoded symbol of degree $\left\lfloor \frac{d}{1-c} \right\rfloor$, subject to the maximum degree.

Plots showing the performance of approximate reconciliations trees are omitted, as their accuracy is similar to that of Bloom filters (but the search time can be superior, as described earlier).

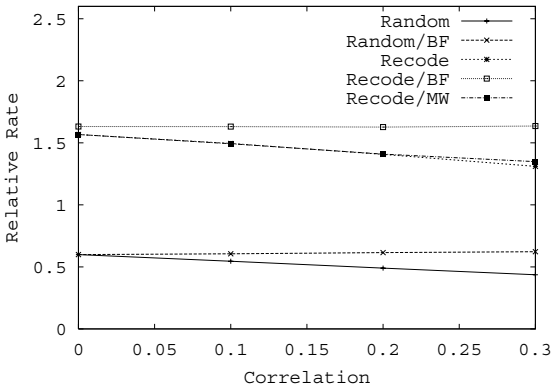
6.3 Evaluation

Our first set of experiments examines the performance of various strategies in choosing which content to send over a particular connection between two peers. Figure 5 shows the overhead as correlation varies, where the overhead is the *additional* overhead, beyond that of a baseline transfer in which encoded content is used. Figure 5(a) features a "compact" scenario with only $1.1n$ distinct symbols in the system, only slightly more than necessary for recovery. Figure 5(b) features a "stretched" scenario with $1.5n$ distinct symbols in the system.

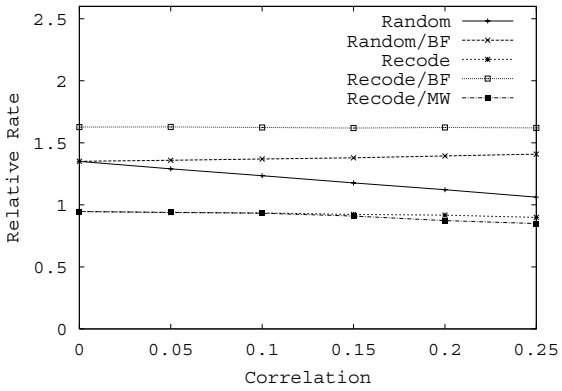
The compact scenarios correspond to situations where useful symbols are scarce and receivers must carefully collect all possible symbols from sending peers in order to recover the entire file. Compact scenarios are comparable to those where exact set reconciliation is important when the content is not encoded. The stretched scenarios are more optimistic, since receivers have more freedom in which symbols to obtain. In these scenarios, receivers may place more preference on higher-throughput senders since correlation of content is less problematic.

In Figure 5, the receiver is initially in possession of half of the distinct symbols in the system. The sender stores the other half of symbols plus a fraction of the receiver's symbols to achieve the specified level of correlation. In all figures, no nodes with partial content initially have more than n symbols, resulting in restricted ranges of correlation.

In the compact scenario, the random selection strategy performs very poorly - this strategy is precisely characterized by the well known Coupon Collector's problem [14]. When exactly n symbols are present in the system, random selection requires $O(\log n)$ symbols on average to recover each *useful* symbol, so this strategy is not suitable for sending all of a large set of symbols. The recoding

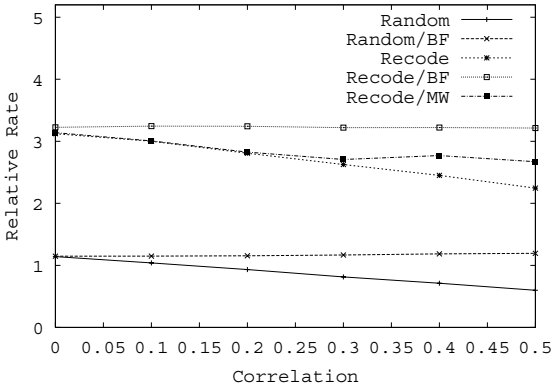


(a) Compact Scenarios, $1.1n$ distinct symbols in system

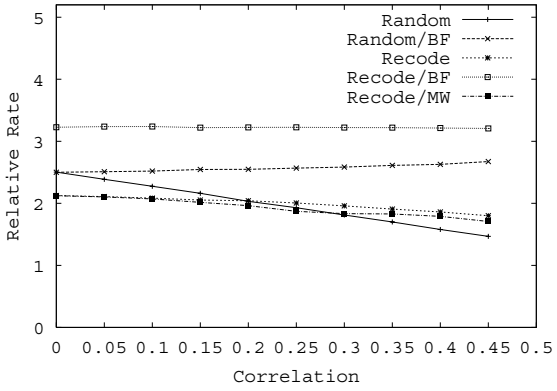


(b) Stretched Scenarios, $1.5n$ distinct symbols in system

Figure 7: Relative transfer rates using two partial senders, compared with a single full sender.



(a) Compact Scenarios, $1.1n$ distinct symbols in system



(b) Stretched Scenarios, $1.5n$ distinct symbols in system

Figure 8: Relative transfer rates using four partial senders, compared with a single full sender.

strategies perform consistently better in these experiments. Recoding with Bloom filters consistently is the best in these experiments, maintaining a low constant overhead as correlation increases. Recoding without any summary information starts similarly but performs poorly under high correlation, while the performance of recoding with minwise summaries degrades at about half its rate.

In the stretched scenario, the random selection strategy performs much better since only O(1) symbols are needed on average before each useful symbol is recovered. Simultaneously, the more oblivious recoding strategies perform much worse since they recode over too large a domain of regular symbols.

The plots of Figure 6 show the speedup in effective download rates when a sender with partial content is added alongside a full sender. The scenarios are like those of Figure 5 with the addition of a full sender. The full sender sends regular symbols at the same rate that the partial sender sends recoded symbols. The speedup is relative to the rate that regular symbols are received from the full sender.

Additionally, these experiments illustrate the importance of choosing a strategy based on correlation, the amount of available content and the presence of full senders. As in the previous experiments, strategies with Bloom filters out-perform their oblivious counterparts. The random selection strategies also perform well, since the full sender provides additional symbols into the system, moving away from compact scenarios. The more oblivious recoding strategies (minwise and no summary) perform poorly here since they encode over too large a domain of regular symbols.

Figures 7 and 8 show the results of downloading from multiple partial senders in a parallel fashion. As with the previous experiments, the speedups shown are relative to downloading from a single full sender. In these experiments, each of the symbols in the system is initially either distributed to all of the peers or is known to only one peer. Each peer in the system initially has the same number of symbols. While not as efficient as full senders, these flows are additive as with a true digital fountain.

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 represent-

ing 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 at most a handful of packet payloads 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] Altavista. www.altavista.com.

[2] ANDERSEN, D., BALAKRISHNAN, H., KAASHOEK, F., AND MORRIS, R. Resilient overlay networks. In *Proceedings of the 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] 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., LUBY, M., AND MITZENMACHER, M. Accessing multiple mirror sites in parallel: Using tornado codes to speed up downloads. In *Proceedings of IEEE INFOCOM* (March 1999), pp. 275–83.

[7] BYERS, J. W., LUBY, M., MITZENMACHER, M., AND REGE, A. A digital fountain approach to reliable distribution of bulk data. In *Proceedings of ACM SIGCOMM* (Vancouver, September 1998).

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

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

[10] DABEK, F., KAASHOEK, F., KARGER, D., MORRIS, R., AND STOICA, I. Wide-area cooperative storage with cfs. In *Proceedings of the ACM Symposium on Operating Systems Principles* (Banff, Canada, October 2001).

[11] FAN, L., CAO, P., ALMEIDA, J., AND BRODER, A. Summary cache: A scalable wide-area cache sharing protocol. In *SIGCOMM Symposium on Communications Architectures and Protocols* (1998), pp. 245–265.

[12] Fasttrack P2P stack. http://www.fasttrack.nu/c_portfolio.html.

[13] JANNOTTI, J., GIFFORD, D., JOHNSON, K., KAASHOEK, M., AND O'TOOLE, J. Overcast: Reliable multicasting with an overlay network. In *Proceedings of OSDI* (San Diego, California, 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 *SIGCOMM Symposium on Communications Architectures and Protocols* (September 1997).

[16] LUBY, M., MITZENMACHER, M., SHOKROLLAHI, A., SPIELMAN, D., AND STEMANN, V. Practical loss-resilient codes. In *Proceedings of the* 29^{th} *ACM Symposium on Theory of Computing* (April 1997).

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

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

[19] MINSKY, Y., TRACHTENBERG, A., AND ZIPPEL, R. Set reconciliation with nearly optimal communication complexity. In *ISIT* (Washington, DC, June 2001).

[20] Mojonation. http://www.mojonation.net.

[21] NONNENMACHER, J., BIERSACK, E., AND TOWSLEY, D. Parity-based loss recovery for reliable multicast transmission. In *Proceedings of ACM SIGCOMM* (September 1997).

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

[23] RATNASAMY, S., FRANCIS, P., HANDLEY, M., KARP, R., AND SHENKER, S. A scalable content-addressable network. In *SIGCOMM Symposium on Communications Architectures and Protocols* (San Diego, CA, August 2001).

[24] RIZZO, L. Effective erasure codes for reliable computing. In *Computer Communication Review* (April 1997).

[25] RODRIGUEZ, P., KIRPAL, A., AND BIERSACK, E. W. Parallel-access for Mirror Sites in the Internet. In *Proceedings of IEEE INFOCOM* (March 2000), pp. 864–873.

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

[27] SAVAGE, S., COLLINS, A., HOFFMAN, E., SNELL, J., AND ANDERSON, T. The End-to-End Effects of Internet Path Selection. In *SIGCOMM Symposium on Communications Architectures and Protocols* (August 1999).

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

[29] ZHAO, B., KUBIATOWICZ, J., AND JOSEPH, A. Tapestry: An infrastructure for fault resilient wide-area location and routing. Tech. Rep. UCB//CSD-01-1141, U. C. Berkeley, April 2001.