Bandwidth Efficient String Reconciliation using Puzzles

Sachin Agarwal

Deutsche Telekom AG, Laboratories

Ernst-reuter-platz 7

Berlin 10587, Germany

Email: sachin.agarwal@telekom.de

Vikas Chauhan

Email: chauhan@alum.bu.edu

Ari Trachtenberg

Boston University

Boston 02215, MA, USA

Email: trachten@bu.edu

A version of this work will appear in the IEEE Transactions on Parallel and Distributed Systems, 2006.

This work was completed while Sachin Agarwal was at Boston University. This research is based, in part, upon work supported by the National Science Foundation under Grants CCR-0133521 and ANI-0240333.

Abstract

Of considerable interest in recent years has been the problem of exchanging correlated data with minimum communication. We thus consider the problem of exchanging two similar strings held by different hosts. Our approach involves transforming a string into a multi-set of substrings that are reconciled efficiently using known multi-set reconciliation algorithms, and then put back together on a remote host using tools from graph theory. We present analyses, experiments and results to show that the communication complexity of our approach for high-entropy data compares favorably to existing algorithms including *rsync*, a widely-used string reconciliation engine. We also quantify the tradeoff between communication and the computation complexity of our approach.

Index Terms

efficient file synchronization, string reconstruction, rsync

I. Introduction

We address the problem of reconciling similar strings held by two distinct hosts. The strings may represent files, web pages, or genomic code residing with similar content on different hosts. Formally, we consider two distinct hosts A and B, each with a string σ_A and σ_B respectively, composed over some alphabet Σ. The *efficient string reconciliation* problem is thus for host A to determine σ_B and for host B to determine σ_A with minimum communication.

The problem of string reconciliation appears in a number of applications, most prominently in file synchronization. In network applications, where frequent updates are made to copies of the same file, string reconciliation can be used to share the updates among networked computers. For example, it can be used to reconcile document replicas in replicated file systems, such as Ficus [16], Coda [28] and Echo [2]. Data distribution systems can leverage the similarity between the current and earlier versions of data to transmit updates without disrupting normal communication traffic [23]. Mirroring of

internet web servers also requires the repeated and efficient reconciliation of different versions of the same files. Finally, image reconciliation can be thought of as a two dimensional generalization of string reconciliation, and, in general, string reconciliation algorithms can be used as a basis for reconciling various types of structured data.

A. Approach

The idea behind our approach to string reconciliation is to divide each string into a multi-set of "puzzle pieces". These multi-sets are reconciled using CPIsync [1, 20, 28], an efficient distributed algorithm for set and multi-set reconciliation. In the final step these "puzzle pieces" comprising the multi-set are put together, like a puzzle, in order to form the original string data. We show that the communication complexity of our proposed approach scales linearly in *edit distance* between reconciling strings, rather than being linear in string length as is the case for conventional reconcilers such as rsync [32]. Recall that the edit distance between two strings is the minimum number of edit operations (insertion, deletion or replacement of single characters) required to transform one string into the other, and this quantity can be significantly smaller than the lengths of either string.

Puzzle pieces are constructed out of proximate characters of the original string by repeatedly applying a mask. Formally, a *mask* is a binary array; applying a mask to a string involves computing a dot product of the mask with a substring of the string. This process is known as 'shingling' [3, 4] in the special case when the mask is comprised of all ones. In other words, applying a mask involves placing it over the string, beginning at a certain character, and reading off all characters corresponding to 1 bits in the mask, thus producing one *puzzle piece*. To divide a string into pieces, one simply applies the mask at all shifts in the string ($i.e.$ starting at each character).

As a concrete example, consider the string 01010010011 under the mask 111, which has length $l_m = 3$. We artificially pad the string with anchors (*i.e.,* characters not in the string alphabet) at the

beginning and end to get $\$01010010011\$$, which is subdivided into the following multi-set of puzzle pieces:

$$\{\$01, 010, 101, 010, 100, 001, 010, 100, 001, 011, 11\$\}.$$

The key observation of our approach is that a string edit only affects a small number of puzzle pieces that are within the local vicinity of the edit. As such, though strings may have many pieces, these pieces will be mostly the same as long as the strings are mostly the same. This, in turn, means that the communication needed to synchronize the strings with CPIsync will be small for strings that are very similar. Once the pieces of a string are known, it can be reconstructed by enumerating Eulerian paths (similarly to [15]). Unfortunately, the number of such paths (and hence the computational burden of the algorithm) can be quite large in the general case. As such we provide analytical tools for picking an appropriate mask length to keep the decoding process feasible

a) Outline: In Section II we briefly discuss some well known string reconciliation bounds proposed by others. In Section III-A we briefly summarize the CPIsync algorithm and then formally describe how a string is represented as a multi-set and reconstructed back at the decoding host. In Sections III-B through III-C we briefly provide some of the necessary graph-theoretic background needed to understand our approach, which is formally described in Section III-D. Thereafter, in Section IV we analyze the proposed approach in terms of communication and computation complexity. We provide an experimental comparison of the proposed approach with the well known open-source rsync [32] incremental file transfer utility in Section VI and present our conclusions in Section VII.

II. Existing Results

The edit distance between two strings is the minimum number of edits (*i.e.,* insertion, deletion or replacement of single characters) required to transform one into the other. Orlitsky [24] presented

some information theoretic bounds on the amount of communication needed for exchanging documents modelled as random variables with a known joint distribution. He also proposed an efficient one-way protocol for reconciling documents that differ by at most α string edits. In Orlitsky's model, host P_X holds string X which differs from the string Y held by P_Y by at most α edits. It is shown that host P_X requires at least

$$\alpha \log |X| - o(\alpha \log |X|) \text{ bits}$$

of communication to communicate string X to host P_Y. Orlitsky and Viswanathan [25] also present an efficient protocol for transmitting a string X to another host with a string Y, an edit distance k from X. This protocol succeeds with probability $1 - \epsilon$ and requires the communication of at most

$$2k \log |X| \left(\log |X| + \log \log |X| + \log \left(\frac{1}{\epsilon} \right) + \log k \right) \text{ bits.} \tag{1}$$

Cormode, Paterson, Ṣahinalp and Vishkin [9] have also proposed protocols that minimize communication complexity in reconciling similar documents and have come up with bounds on these values based on the Hamming, LZ and Levenshtein metrics. In their protocols, the amount of communication needed to correct an edit distance of d between two strings is upper bounded by

$$4d \log(2n/d) \log(2\hat{d}) + O\left(d \log n \log \frac{\log n}{\ln 1/p_c} \right) \text{ bits,}$$

where n is the length of the string, $\hat{d}$ is the bound on the edit distance and p_c is the desired probability of success. They also show how to identify different pages between two copies of an updated string using error-correcting codes, a problem addressed earlier by Barbara and Lipton and also by Abdel-Ghaffar and El Abbadi using Reed-Solomon codes [18].

More recently, Evfimievski [13] presented a probabilistic algorithm for communicating an edited

binary string over a communication network with arbitrarily low probability of error. The communication complexity is upper bounded by

$$1000k^2 \log|y|(16 + \log|x| + 5\log|y| + \log(\epsilon^{-1}))\text{ bits}, \tag{2}$$

where k is the edit distance, $|x|$ and $|y|$ are the lengths of the strings and ϵ is the error probability. A graphical comparison of these protocols is presented in Section V.

It should be noted that in many cases it is assumed that an original string is maintained and known on both hosts and then synchronized at a later date, so that edits can be recorded and compressed upon reconciliation [9, 31]. Our model, on the other hand, does not presume any a priori information between the hosts. This kind of scenario occurs, for example, if the hosts were given their strings by a third party (*i.e.,* without any history of edits, as occurs in some networked settings), or if the strings were derived by some foreign process that generates related output (*e.g.,* related DNA strings).

III. Theoretical Background

We next review some theoretical underpinnings of our approach, culminating in Section III-D with a formal statement of our proposed string reconciler.

A. Overview of CPIsync

The Characteristic Polynomial Interpolation Synchronization protocol [CPIsync] [19–21] reconciles two sets S_A and S_B using an algebraic approach based on rational function interpolation in a finite field. A salient feature of this protocol is that its communication complexity is broadly determined by the number of differences between reconciling sets, rather than the sizes of the sets.

More precisely, the probabilistic version of this algorithm [21] requires

$$2(b+1)m + b + bm_A + m + k \text{ bits}$$

to reconcile sets S_A and S_B of b-bit vectors with a symmetric difference $m = |S_A \oplus S_B|$, unidirectional differences $m_A = |S_A \setminus S_B|$ and $m_B = |S_B \setminus S_A|$, and a confidence parameter k related exponentially to the expected probability of success.

CPIsync generalizes nicely to multi-set reconciliation and a recent multi-round extension [19] runs in expected linear-time.

B. Concepts

An *Eulerian cycle* is a graph cycle that traverses each edge exactly once. The *de Bruijn digraph* $G_{l_m}(\Sigma)$ over an alphabet Σ and length l_m is defined to contain $|\Sigma|^{l_m-1}$ vertices, each corresponding to a length $(l_m - 1)$ string over the alphabet. An edge from vertex v_i to v_j exists with label l_{ij} if the string associated with v_j contains the last $l_m - 2$ characters of v_i followed by l_{ij}. Thus, each edge (v_i, v_j) represents a length l_m string over Σ defined by the label of v_i followed by l_{ij}. An example of the de Bruijn digraph $G_3(\{0,1\})$, taken from [30], is shown in Fig. 1.

1) Modified de Bruijn Digraph: The following steps transform a de Bruijn digraph $G_{l_m}(\Sigma)$ into a modified de Bruijn digraph similarly to the construction in [30], for a particular multi-set of pieces of a string drawn from alphabet Σ and encoded with a mask of length l_m:

- parallel edges are added to the digraph for each occurrence of a particular piece in the multi-set.
- edges which represent strings not in the multi-set are deleted.
- vertices with degree zero are deleted.
- two new vertices and edges corresponding to the first and last pieces of the encoded string are

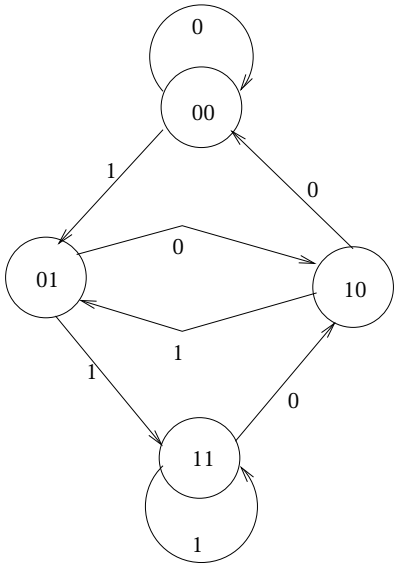

Fig. 1. The de Bruijn digraph of length 3 binary strings.

added.

- An artificial edge is added between the two new vertices to make their in-degree equal their out-degree (*i.e.,* one).

There is a one to one correspondence between the edges in this graph and the pieces in the multi-set except for the artificial edge. The modified de Bruijn digraphs for the strings $\sigma_A = 01010011$ and $\sigma_B = 01010010011$ (after padding with anchors) on hosts A and B are shown in Fig. 2. The problem of determining the original string from a multi-set of pieces was shown to be equivalent to finding the correct Eulerian path in a modified de Bruijn digraph [30].

Note that in our application, the modified de Bruijn digraph necessarily has at least one Eulerian cycle (and typically several) corresponding to the string that it encodes. By virtue of this fact, $in{-}degree\ d_{in}(i)$ of any vertex v_i necessarily equals its $out - degree\ d_{out}(i)$. We may thus define $d_i \hat{=} d_{in}(i) = d_{out}(i)$ and form a diagonal matrix M from the degrees of the n vertices of the graph

$$M = diag(d_1, d_2, d_3, ..., d_n).$$

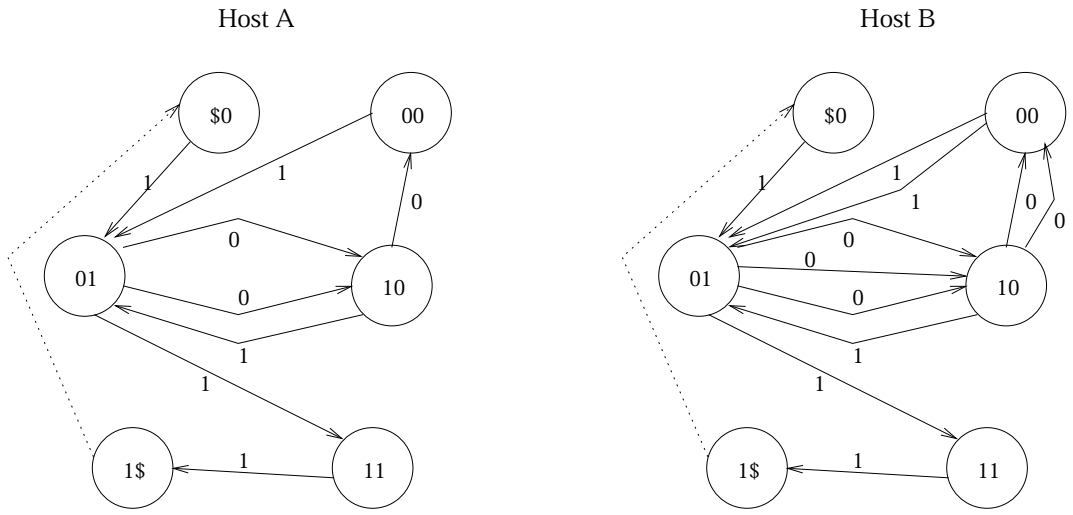

Fig. 2. The modified de Bruijn digraphs for the strings on hosts A and B.

When put together with the adjacency matrix $A = [a_{i,j}]$ of the graph, this produces the Kirchhoff matrix $C = M - A$.

Theorem 1, a modification of the well-known B.E.S.T. theorem [14], provides the number of Eulerian cycles in a modified de Bruijn digraph. The Theorem and its proof were presented in [17], and we restate it for completeness.

Theorem 1 (Modified B.E.S.T.-Theorem)

For a general Eulerian digraph, the total number of Eulerian cycles R is given by

$$R = \frac{\Delta\Pi_i(d_i - 1)!}{\Pi_{ij}a_{ij}!}.$$

C. The Backtrack Algorithm

Theorem 1 gives us the number of Eulerian cycles in a graph. In order to enumerate these Eulerian cycles (corresponding to possible string decodings) we may use the exhaustive Backtrack method [22]. In this method, a path is greedily "grown" into an Eulerian cycle; at each step, the partial path is examined to see if it could lead to an Eulerian cycle, with the algorithm backtracking to a previous

vertex if this examination fails. Whenever the path length equals the number of edges in the graph, the path is output and the algorithm backtracks to the previous vertex in search of another cycle.

D. STRING-RECON

We now have the necessary tools to describe our string reconciliation protocol (STRING-RECON).

Consider two hosts A and B holding strings σ_A and σ_B respectively. The mask-length l_m is pre-determined on the basis of the analysis presented in Section IV-B. Host A determines σ_B and host B determines σ_A as follows:

1) Host A transforms string σ_A into a multi-set of pieces MS_A using a mask-length of l_m and constructs a modified de Bruijn digraph from the pieces. Host B transforms string σ_B into MS_B using the same mask-length l_m and constructs a modified de Bruijn digraph from the pieces.
2) A and B determine the index of the desired decoding in the sequential enumeration of all Eulerian cycles in the graph. Thus, n_A corresponds to the index of the Eulerian cycle producing string σ_A, and similarly n_B corresponds to σ_B.
3) A and B transform multi-sets MS_A and MS_B into sets with unique (numerical) elements S_A and S_B by concatenating each element in the set with the number of times it occurs in the multi-set and hashing the result; the sets S_A and S_B store the resulting hashes. Any one way hash function with a low collision probability ($e.g.$ MD5) can be used; however, a higher number of bits for the hash function would result in a higher overall communication complexity.
4) The CPIsync algorithm is executed to reconcile the sets S_A and S_B. In addition, A sends n_A to B and B sends n_B to A. At the end of this step, both A and B know S_A, S_B, n_A and n_B. Host A then sends elements corresponding to hashes in the set $S_A \setminus S_B$ to B and B sends elements corresponding to the hashes in the set $S_B \setminus S_A$ to A.

5) A and B construct the multi-sets MS_B and MS_A respectively from the information obtained in the previous step. They also generate the corresponding modified de Bruijn digraphs.
6) The decoding algorithm is applied by A and B to determine σ_B and σ_A respectively.

IV. Analysis

There are two important measures of the efficiency of the proposed string reconciliation protocol. One is the communication complexity (*i.e.,* the number of bytes exchanged) and the other is the computation complexity of the algorithm running on the hosts. Communication complexity depends on the number of differences in the shingle-sets that CPIsync reconciles. The computation complexity, on the other hand, is determined by the number of possible Eulerian cycles that the Backtrack algorithm finds in the modified de Bruijn graph. In this section we provide analytical bounds on the communication complexity and show how to reduce the number of Eulerian cycles by increasing mask-length. We then analytically show the tradeoff between these two measures.

A. String Edits and Communication Complexity

Suppose hosts A and B initially have strings σ_A and σ_B respectively which are of length n. The hosts use the same mask of length l_m to generate piece multi-sets. The following results bound the number of differences Δ_{AB} that can occur between the piece multi-sets of A and B in the face of certain edits.

Theorem 2

If two strings σ_A and σ_B differ by d non-degenerate edits[1] *then the number of differences Δ_{AB} between resulting piece multi-sets is bounded by*

$$d + 2(l_m - 1) \leq \Delta_{AB} \leq min((2l_m - 1)d, 2(n - l_m + 1) + d).$$

[1] *We assume the non-degenerate case where edits occur after the l_m-th character and before the l_m-th character.*

Proof: The best case would be achieved when the d insertions occur at a single location. In this case, the host that has the augmented string would have $d + l_m - 1$ different pieces. The other host will have $l_m - 1$ different pieces, totalling up to $d + 2(l_m - 1)$ differences. The worst case would occur when the insertions are spaced a distance l_m apart. One such insertion would cause a difference of $2l_m - 1$ pieces. d such insertions would cause a difference of $d(2l_m - 1)$ pieces. When the insertions (so spaced) span the entire string, all the pieces are different on both the strings and thus the difference becomes $2(n - l_m + 1) + d$. Note that in this case, $(2l_m - 1)d$ exceeds $2(n - l_m + 1) + d$ and thus, we have to resort to the min function for a tighter bound in the general case.

Deletion on one host can be considered as insertion on the other and replacements can be considered to be an insertion followed by a deletion, thus providing analogous results to Theorem 2. ■

Corollary 1

If d edits are made on σ_B, then the upper bound on the number of symmetrical differences between the multi-sets A and B can always be expressed as

$$\Delta_{AB} \leq 2l_m d.$$

The proof of Corollary 1, showing that the number of elements in the multi-sets affected by such edits is linear in the number of edits performed, follows readily by induction on the number of edits.

The following communication bound is also a natural consequence of Theorem 2.

Corollary 2

The amount of communication used by STRING-RECON to reconcile binary strings differing in e edits

using masks of length l_m is bounded by

$$\textit{COMM} \;\leq\; 2(b+1)m + b + bm_A + m + k + ml_m + \log(R_1R_2)\textit{bits}$$

where the symbols have the same meaning as in Section III-A and R_1 and R_2 are the total number of Eulerian cycles for the modified de Bruijn digraphs at hosts A and B respectively.

B. Mask-length and maximum graph node degree

The degrees d_i form a dominant term in the computation of the number of Eulerian cycles in a graph (Theorem 1). In order to keep the number of Eulerian cycles low, the maximum degree $max(d_i)$ should be as small as possible, ideally 1. We now formalize the intuitive idea that increasing the mask-length reduces the maximum degree of a Eulerian graph, and hence reduces the general decoding complexity of the decoding algorithm of Section III-C, at the cost of increased communication complexity of (2). Our analysis below is based on a simplified model of strings as random sequences of bits, with each bit being independent and identically distributed (iid), and occuring as 1 with probability p. Though the results can be simply generated to random sequences of k-ary characters, a full analysis for arbitrarily modeled strings is left as an open problem.

Theorem 3

Consider the de Bruijn graph of a random binary string σ. If a mask of length l_m is used to construct the de Bruijn graph then the expected degree $d^{(k)}$ of a node whose label has Hamming weight k is given by

$$d^{(k)} = (n - l_m + 1)p^{l_m-k-1}(1-p)^k. \tag{3}$$

Proof: If we select any $l_m - 1$ consecutive locations in the original string that comprise a node label in the de Bruijn graph, the probability of the this $l_m - 1$ bit sequence being some particular sequence

of weight k is $p^k(1-p)^{l_m-k-1}$. This l_m-1 bit node label has out-going edges labeled 0 and 1. The probability of an l_m bit sequence with the first l_m-1 bits being the node label and the last bit being a 0 or a 1 in the original string is given by

$$p^k(1-p)^{l_m-k} + p^{k+1}(1-p)^{l_m-k-1} = p^{l_m-k-1}(1-p)^k$$

The original string is divided into $n-l_m+1$ pieces using the l_m bit mask. By the linearity of expectation, and given that all bits of the string are *iid*, the out-degree of a node with a label of Hamming weight k $(0 \leq k \leq l_m-1)$ we obtain the expression for the node degree in (3). ■

Theorem 3 enables us to compute in expectation the maximum node degree in the de Bruijn graph, giving a rough idea of the number of Eulerian cycles in Theorem 1. This follows from the fact that if all the graph's nodes have degree one then there is exactly one Eulerian cycle in the graph. The degree $d^{(k)}$ in (3) is a monotonically decreasing function of the mask-length and for a sufficiently long mask-lengths the maximum degree of every node in the graph will become one. Intuitively, by increasing the mask-length l_m we are introducing more and more *distinct* nodes in the de Bruijn graph and reducing duplicate edges between nodes, hence decreasing the number of Eulerian cycles.

Theorem 4

The mask-length l_m required to reduce to unity the expected maximum out-degree of a de Bruijn graph of a random binary string σ with bit probability $p \geq 0.5$ is

$$l_m = \frac{n\ln(p) + \ln(p) + W(-\ln(p)e^{-n\ln(p)})}{\ln(p)} \tag{4}$$

Where W is the LambertW function [8] that satisfies

$$W(x)e^{W(x)} = x$$

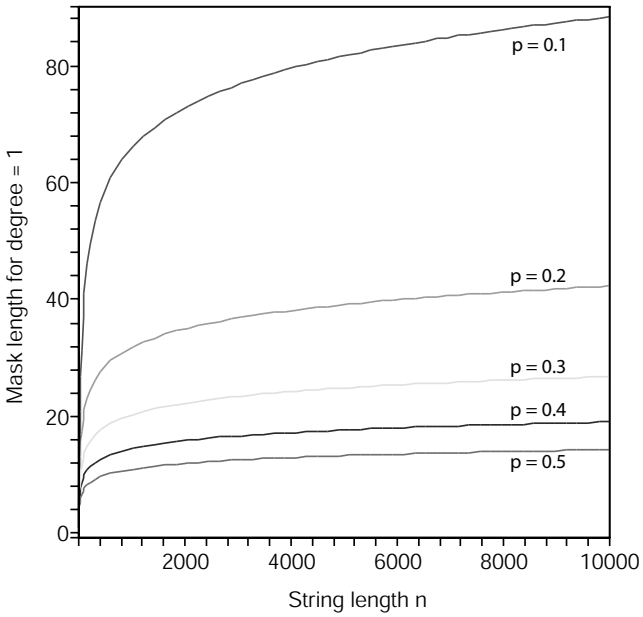

Fig. 3. String length n versus Mask-length l_m for the maximum degree in the De Bruijn graph to be one.

Proof: For $p \geq 0.5$ the node(s) with maximum out-degree $d^{(k)}$ correspond to those that have $k = 0$ in (3), giving

$$d^{(0)} = (n - l_m + 1)p^{l_m - 1}. = 1 \tag{5}$$

Solving (5) for out-degree equal to one we get the expression in (4). ■

A similar result can be obtained for the case of $p < 0.5$ by replacing p by $1 - p$ in (4).

Corollary 3

The mask length required to maintain the maximum degree at unity grows at most logarithmically with n, the length of the bitwise random binary string σ.

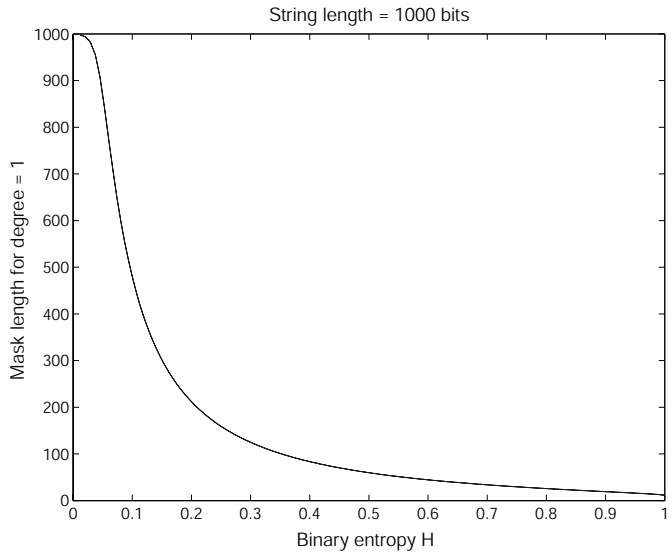

Fig. 4. Binary entropy of a string bit versus Mask-length l_m in order for the expected maximum degree in the de Bruijn graph to be one.

Proof: To see this we observe that the lambertW function W(z) can be expanded as in [8]

$$W(z) = L_1 - L_2 + \frac{L_2}{L_1} + \frac{L_2(-2+L_2)}{2{L_1}^2} + \frac{L_2(6-9L_2+2{L_2}^2)}{6{L_1}^3} + \frac{L_2(-12+36L_2-22{L_2}^2+3{L_2}^3)}{12{L_1}^4} + O\left(\left\{\frac{L_2}{L_1}\right\}^5\right)$$

where $L1 = ln(z)$ and $L2 = ln(ln(z))$ and $ln(\cdot)$ is the natural log function. Noting that $\frac{L2}{L1} \leq 1$ we can interpret the above LambertW expansion as

$$W(z) \leq L_1 - L_2 + c\left(\frac{L_2}{L_1}\right), \tag{6}$$

where c is a constant. Substituting (6) into the expression for the mask-length in (4) and taking the limit as n approaches ∞, we get $l_m = O(ln(n))$ ■

Figure 4 shows the mask-length needed to ensure an expected single decoding cycle (from 4) as a function of the string's bit-entropy $H(p)$. Clearly, as the value of p approaches 0, the required mask-length will approach $n - l_m + 1$. On the other hand, when the entropy of the original string is high

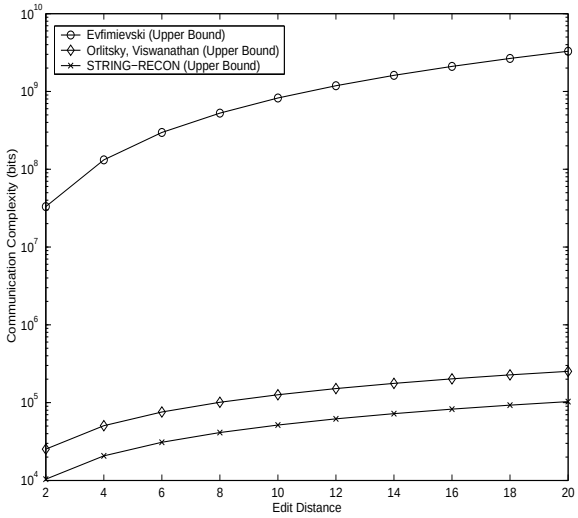

Fig. 5. Upper bound on the communication complexity for STRING-RECON compared to the theoretical upper bounds given in [13] and [25]

(*i.e.,* $p \to 0.5$) it is more *unlikely* to find repeating l_m bit sequences in the string that would make the corresponding node degrees higher in the modified de Bruijn graph, resulting in a relatively small mask length.

While the results hold for uniformly distributed random binary strings (and can be generalized for uniformly distributed *k-ary* strings), we leave as open the question of bounding the number of Eulerian cycles in strings from more complicated models (*e.g.,* natural English text). In our experiments we found that suitably long mask-lengths have the same desirous effect on English language strings of reducing the number of Eulerian cycles to a small number (usually 1), although the required increase in mask-length significantly impacts the communication size.

V. Comparison with Existing Results

The theoretical communication complexity upper bound of STRING-RECON was compared to the theoretical upper bound by Evfimievski [13] and the theoretical upper bound by Orlitsky and Viswanathan [25]. The results are summarized in Figure 5. The string on both hosts was an English language string

of size 1000 characters, encoded with 8 bits per character. An edit distance of k between two English language strings would, in general, translate to an edit distance of at least k in the binary representations of the strings.

We used a mask-length of 11 corresponding to length predicted in Equation (4) for unique decoding of the modified de Bruijn digraph. In order to put all methods on an equal footing, we set the error probability ϵ to the error probability of CPIsync:

$$\epsilon = m \left(\frac{|S_A| + |S_B| - 1}{2^b} \right)^k,$$

with the confidence parameter $k = 3$ set arbitrarily. Note that Evfimievski's protocol in [13] and Orlitsky and Viswanathan's protocol in [25] are only for one-way reconciliation, and we have adapted them to make a fair comparison to STRING-RECON, which performs a full reconciliation of the strings.

Though the STRING-RECON upper bound appears the best, all the bounds are still clearly weak and not necessarily representative of actual performance, which is much harder to quantify.

VI. Experiments and Comparison to rsync

In the first set of experiments we implemented the proposed scheme and studied the communication performance for randomly generated binary strings. In Figure 6 we see that the communication complexity grows linearly with the number of edits since CPIsync requires more rational function evaluations. For a constant number of uniformly distributed random edits, the communication grows logarithmically with the string length because the mask-length has to be be increased logarithmically with string length in order to expect only one Eulerian cycle.

We next compared our algorithm to the popular rsync [32] utility, an open-source incremental file-transfer program. In the most common setting host A wants to copy string σ_A onto host B that already has a similar string σ_B. Rsync works as follows: host B first divides string σ_B into S byte disjoint

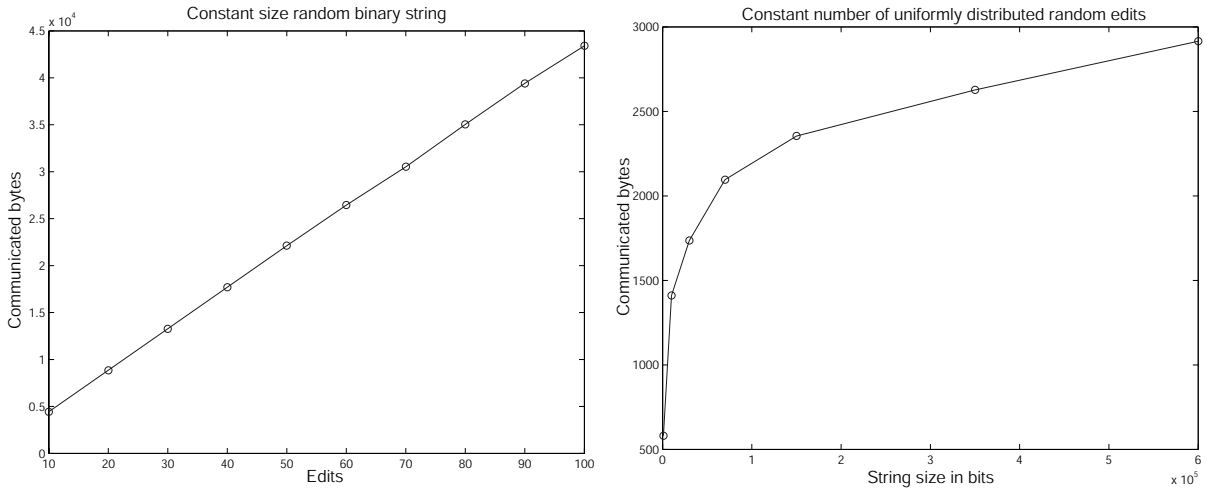

Fig. 6. Communication performance of the proposed scheme over random binary bit-strings and uniformly distributed edits

blocks and sends a strong 128-bit MD4 [27] checksum as well as a weaker 32-bit rolling checksum of each block to Host A. Host A then determines the blocks at all offsets that match the checksums and determines string σ_A's data and block indices that need to be sent to host B for the latter to reconstruct string σ_A. If σ_A and σ_B are very similar then the overhead of sending the hashes is more than offset by the savings of only sending the literal data corresponding to the differing parts. Note that sending hashes of disjoint blocks corresponds to communication complexity that is linear in the length of the input string.

Our input data included varying length snippets of the text of Winston Churchill's 'Their finest hour' speech [6] over the English alphabet with punctuation. We introduced 'edit bursts' of changes in order to mimic a human editing the text of the speech. Each edit burst was 5 characters long and placed randomly in the text. We computed the number of bytes communicated by the proposed scheme based on the communication bound of Corollary 2 for non-binary alphabets and explicitly verified that the running time was small for the chosen mask length by decoding and enumerating all the Eulerian cycles using the backtrack algorithm discussed in Section III-C.

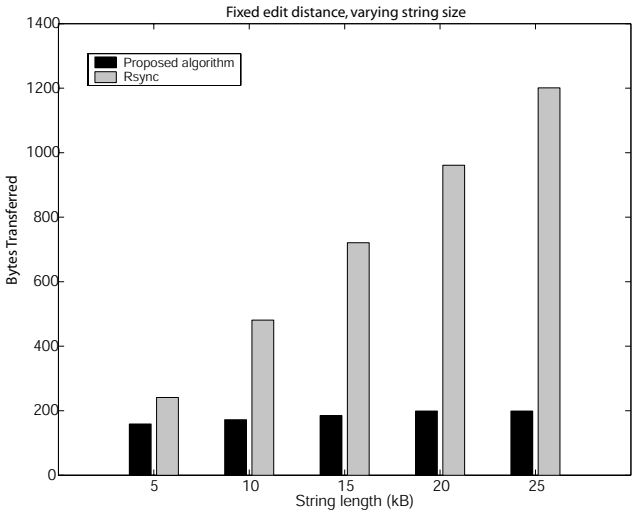

Fig. 7. Comparison to rsync; The input string was snippets of varying length taken from Winston Churchill's *'Their finest hour'* speech. There was one edit burst that introduced an edit distance of 5 in the text.

In Figure 7 we show the comparison of the proposed scheme with rsync. As expected the communication complexity of the proposed scheme grows very slowly with increasing string length for a fixed number of differences. Note that the communication complexity could have been decreased substantially by reducing the mask-length, although at the cost of higher running time corresponding to more Eulerian cycles in the de Bruijn graph.

Rsync performed well for larger number of edits as Figure 8 illustrates. The text of Shakespeare's famous play 'Hamlet' (About 175 kB) was used in this experiment. The communication complexity of the proposed scheme grows linearly with the number of differences. This is also true in the case of rsync because differing text has to be transmitted from one host to the other irrespective of any algorithm. The difference is primarily in the order constant.

Table I compares the performance of the proposed algorithm with rsync on various strings for a fixed edit distance. The proposed algorithm performed well for higher entropy strings but communicated more bytes as compared to rsync while reconciling lower entropy strings such as human genome data. Bitorrent's Python source code and the Contig NT-079585.2 (from Human Y chromosome) strings were

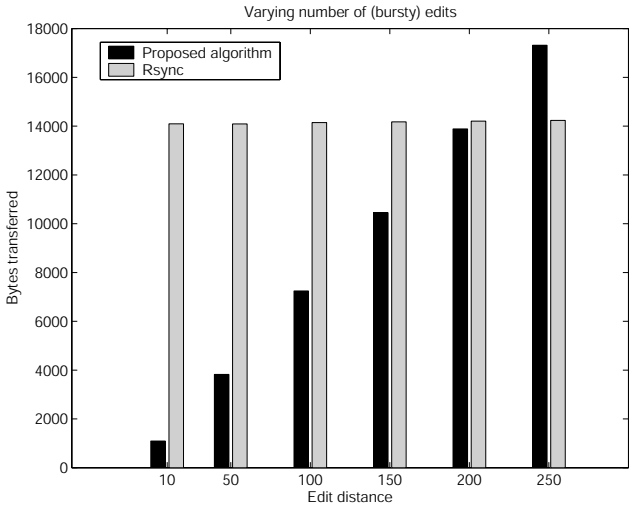

Fig. 8. Comparison to rsync; The input string was of constant length (Shakespeare's play *'Hamlet'*). Each edit burst added an edit distance of 10 in the text.

String description	String length	Proposed algorithm	Rsync
60,000 lexicographically ordered English words	523,373	6473	8,759
Concatenated string of MD5 hashes	380,834	4,563	6,299
Java source code samples from [11]	371,330	5,234	6,456
C++ source code samples from [12]	318,254	4,470	5,820
Bitorrent's Python source code [7]	270,880	4,234	5,382
Contig NT-079585.2 of Human Y chromosome [5]	263,614	8,244	4,999
HTML (300 Google results for 'Turing machine')	250,556	4,006	5,052
Tex source of Shannon's seminal 1948 paper [29]	176,048	917	3,692
HTML (RFCs 793 [10] and 768 [26])	151,302	2,262	3,097

TABLE I

COMPARISON (IN BYTES COMMUNICATED) OF THE PROPOSED ALGORITHM WITH RSYNC FOR VARIOUS TYPES OF STRINGS

of approximately of the same length, but the proposed approach communicated lesser bytes in the former case because of higher entropy as compared to the genomic string in the latter case. Rsync's communication complexity increased linearly with the size of the input string, irrespective of the entropy of the strings being reconciled. We found our scheme outperformed rsync for many other common strings such as English text, source code, and HTML.

In general, our approach works best for strings with a relatively small number of string differences and

relatively high entropy. The results in Section IV-B imply a communication complexity of $O(m \log^2(n))$ bits, which compares favorably to the $\Theta(n)$ communication of rsync.

VII. Conclusions and Future Work

We have proposed a novel string reconciliation algorithm that efficiently reconciles strings differing by a relatively small number of edits. The key to our proposed solution was to effectively translate the string reconciliation problem to a comparable set reconciliation problem, for which efficient algorithms exist. As such, our reconciliation algorithm trades communication efficiency with computation efficiency, allowing it to adapt to a variety of network conditions in a heterogeneous system.

Another feature of the protocol was that it can be implemented with just few transmission rounds of communication, thereby avoiding latency overheads that arise from interaction in protocols such as rsync. This feature is extremely useful while reconciling strings over high latency links such as deep space communication. Our analysis showed that computation costs can be limited to the computational needs of set reconciliation while communication costs grow logarithmically in string size even when the algorithm runs in 'computation-saving' mode by using increased mask-lengths. We showed through several experiments that this can be significantly better than existing string-reconciliation programs whose communication needs grow linearly with string size.

We believe that the proposed algorithm is particularly well-suited for a host of applications where large high entropy strings are changed slowly but asynchronously, for example application mirroring compressed data on the web, peer-to-peer file maintenance, or incremental software updates. It should also be suitable for weakly-connected or high-latency systems, such as deep-space communication where interaction is not feasible.

ACKNOWLEDGEMENTS

The authors would like to thank Prof. David Starobinski for useful comments.

REFERENCES

[1] S. Agarwal, D. Starobinski, and A. Trachtenberg. On the scalability of data synchronization protocols for PDAs and mobile devices. *IEEE Network*, 16(4):22–28, July/August 2002.

[2] A. D. Birrell, A. Hisgen, C. Jerian, T. Mann, and G. Swart. The Echo distributed file system. Technical Report 111, Palo Alto, CA, USA, 10 1993.

[3] A. Z. Broder. On the resemblance and containment of documents. In *SEQS: Sequences '91*, 1998.

[4] A. Z. Broder, M. Charikar, A. M. Frieze, and M. Mitzenmacher. Min-wise independent permutations. *Journal of Computer and System Sciences*, 60(3):630–659, 2000.

[5] National center for biotechnology information. Ncbi human genome resources. http://www.ncbi.nlm.nih.gov/genome/guide/human/.

[6] W. Churchill. Their finest hour. http://www.winstonchurchill.org/i4a/pages/index.cfm?pageid=418.

[7] B. Cohen. Bittorrent. http://www.bittorrent.com/.

[8] R. M. Corless, G. H. Gonnet, D. E. G. Hare, D. J. Jeffrey, and D. E. Knuth. On the lambertw function. *Advances in Computational Mathematics*, 5:329–359, 1996.

[9] G. Cormode, M. Paterson, S. Cenk Sahinalp, and Uzi Vishkin. Communication complexity of document exchange. In *Symposium on Discrete Algorithms*, pages 197–206, 2000.

[10] Darpa Internet Program. Rfc 793: Transmission control protocol. http://www.faqs.org/rfcs/rfc793.html.

[11] B. Eckel. *Thinking in Java*. MindView, Inc, 1. edition, 1998.

[12] B. Eckel. *Thinking in C++*, volume 2. MindView, Inc, 2. edition, 2001.

[13] A. V. Evfimievski. A probabilistic algorithm for updating files over a communication link. In *Ninth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 300–305, 1998.

[14] H. Fleischner. *Eulerian Graphs and Related Topics, Part 1*, volume 1,2. Elsevier Science Publishers B.V., 1991.

[15] A. Guénoche. Can we recover a sequence, just knowing all its subsequences of given length? *Computer Applications in the Biosciences*, 8(6):569–574, 1992.

[16] R. G. Guy, J. S. Heidemann, W. M., T. W. Page, Jr., G. J. Popek, and D. Rothmeir. Implementation of the Ficus Replicated File System. In *USENIX Conference Proceedings*, pages 63–71, Anaheim, CA, June 1990. USENIX.

[17] B. H., H. X., and S. Zhang. Compositional representation of protein sequences and the number of Eulerian loops. *http://arxiv.org/pdf/physics/0103028*, v.1, 2001.

[18] F. J. MacWilliams and N. J. A. Sloane. *The theory of error-correcting codes*. Elsevier, North-Holl, 1977.

[19] Y. Minsky and A. Trachtenberg. Scalable set reconciliation. In *Proc. 40-th Allerton Conference on Comm., Control, and Computing*, Monticello, IL., October 2002.

[20] Y. Minsky, A. Trachtenberg, and R. Zippel. Set reconciliation with nearly optimal communication complexity. In *International Symposium on Information Theory*, page 232, June 2001.

[21] Y. Minsky, A. Trachtenberg, and R. Zippel. Set reconciliation with nearly optimal communication complexity. *IEEE Trans. on Info. Theory*, September 2003.

[22] A. Nijenhuis and H. S. Wilf. *Combinatorial Algorithms*. Academic Press, 1975.

[23] M. Ohta. Incremental zone transfer in DNS. IETF, August 1996. RFC 1995.

[24] A. Orlitsky. Interactive communication: Balanced distributions, correlated files, and average-case complexity. In *IEEE Symposium on Foundations of Computer Science*, pages 228–238, 1991.

[25] A. Orlitsky and K. Viswanathan. Practical protocols for interactive communication. In *IEEE International Symposium on Information Theory*, page 115, 2001.

[26] J. Postel. Rfc 768: User datagram protocol. http://www.faqs.org/rfcs/rfc768.html.

[27] R. Rivest. RFC 1320 - The MD4 Message-Digest Algorithm. Internet-draft, Massachusetts Institute of Technology, April 1992.

[28] M. Satyanarayanan, J. J. Kistler, P. Kumar, M. E. Okasaki, E. H. Siegel, and D. C. Steere. Coda: A highly available file system for a distributed workstation environment. *IEEE Transactions on Computers*, 39(4):447–459, 1990.

[29] C. E. Shannon. A mathematical theory of communication. *Bell Sys. Tech. J.*, 27:379–423, 623–656, 1948.

[30] S. S. Skiena and G. Sundaram. Reconstructing strings from substrings. *Journal Of Computational Biology*, 2:333–353, 1995.

[31] T. Suel and N. Memon. *Algorithms for Delta Compression and Remote File Synchronization*. Academic Press, August 2002.

[32] A. Tridgell. *Effi cient algorithms for sorting and synchronization*. PhD thesis, The Australian National University, 2000.