Communication complexity of document exchange

Graham Cormode * Mike Paterson † Süleyman Cenk Ṣahinalp ‡
Uzi Vishkin §

Abstract

We address the problem of minimizing the communication involved in the exchange of similar documents. We consider two users, A and B, who hold documents x and y respectively. Neither of the users has any information about the other's document. They exchange messages so that B computes x; it may be required that A compute y as well. Our goal is to design communication protocols with the main objective of minimizing the total number of bits they exchange; other objectives are minimizing the number of rounds and the complexity of internal computations. An important notion which determines the efficiency of the protocols is how one measures the distance between x and y. We consider several metrics for measuring this distance, namely the Hamming metric, the Levenshtein metric (edit distance), and a new LZ metric, which is introduced in this paper. We show how to estimate the distance between x and y using a single message of logarithmic size. For each metric, we present the first communication-efficient protocols, which often match the corresponding lower bounds. In consequence, we obtain error-correcting codes for these error models which correct up to d errors in n characters using $O(d \cdot polylog(n))$ bits. Our most interesting methods use a new transformation from LZ distance to Hamming distance.

1 Introduction

This paper focuses on the communication issues related to managing documents shared by physically separated users. Suppose that two users each have a different version of what was at some unknown point of time in the past the same document. Each version was perhaps obtained by a series of updates to that original document (e.g., by a series of transmissions on the internet and updates along the way). Therefore, it would be reasonable to expect that many parts are similar if not identical. The challenge that this paper addresses is how to let one user know the other version and save transmitting redundant information as much as possible. What makes our approach more interesting is that we need not assume that the updates themselves, or any logs of them, are available; the original document is also not assumed to be available; each user has only their current version.

*Dept of EECS, Case Western Reserve University, Cleveland, OH, and Dept of CS, University of Warwick, Coventry, UK; *grc3@eecs.cwru.edu*

†Dept of CS, University of Warwick, Coventry, UK; *msp@dcs.warwick.ac.uk*

‡Dept of EECS, Case Western Reserve University, Cleveland, OH, and Dept of CS, University of Warwick, Coventry, UK; *cenk@eecs.cwru.edu*

§Dept of ECE and UMIACS, University of Maryland, College Park, MD; *vishkin@umiacs.umd.edu*

Formally, let A and B be two such users holding documents x and y respectively. Neither A nor B has any information about the document held by the other. Their goal is to exchange messages so that B computes x; it may also be required for A to compute y as well. Our objective is to design efficient communication protocols to achieve this goal. We measure the efficiency of a protocol in terms of the total number of bits communicated, as per Yao's model of communication complexity (described in [KN97]). We are also interested in minimizing the number of rounds (changes of direction in communication) as well as minimizing the running time of the internal computations performed by A and B.

A trivial but inefficient way of exchanging documents is for A and B to send x and y to each other in full. On the other hand, if A knew y (in addition to x), A could communicate x to B more efficiently by sending a sequence of operations for converting y to x. In this paper we consider protocols in which the set of operations to convert one string into another is pre-determined between the users. Given the set of operations, the minimum number of operations required to convert x into y defines a corresponding distance (in most cases a metric) between x and y. This distance between x and y imposes a lower bound on the length of the message that A sends to B.

We consider three different distance measures, namely, the Hamming metric, the Levenshtein metric (edit distance), and a new LZ metric that we introduce in this paper. Computing the Hamming differences between two binary strings x and y is equivalent to computing the binary string z, where $z[i]$, the i^{th} bit of z, is equal to $x[i] \oplus y[i]$, and $\oplus$ denotes the exclusive-or operation. An obvious deterministic lower bound on the communication complexity of computing the bitwise exclusive-or of two binary vectors of size n is n bits. Hence, any protocol that computes the Hamming distance correctly for all pairs of strings x and y will need to communicate $\Omega(n)$ bits. In fact, this lower bound holds even if the protocol computes the Hamming distance with any constant probability, as shown by Pang and El Gamal [PG86].

For more general, "balanced" measures [1] of string distance $d(x, y)$, the first protocols for exchanging documents are explored by Orlitsky [Orl91]. These protocols assume a fixed upper bound f on $d(x, y)$ and they require the transmission of $\Omega(f \log n)$ bits for the Hamming distance. If $d(x, y) > f$, then the protocols result in an error; if $d(x, y) = o(f)$ then we show in this paper that too many bits are transmitted. Metzner [Met83, Met91], Abdel-Ghaffar and El Abbadi [AGE94] and Barbará and Lipton [BL91] consider the concrete problem of identifying different pages between two copies of an updated file. It is assumed that the differences between pages are aligned with the page boundaries, effectively resulting in Hamming differences. They also assume a known upper bound f on the number of pages that are different, as per the framework in [Orl91]. In particular [AGE94] describes a protocol based on error-correcting codes; this protocol sends a single message of $O(f \log n)$ bits to correct up to f differences.

Contributions. Our most interesting protocols involve only two rounds of communication. The first round involves estimating $d(x, y)$. Here B computes and sends A a $(\log |y|)$-bit random signature of y. This signature enables A to compute an estimate $\hat{d}(x, y)$ for $d(x, y)$. For the Hamming metric this signature is computed by hashing a sequence of random bit

[1] Balanced measures include the ones which are symmetric, such as the edit distance.

samples from y of increasing size (see Section 4.1). For the LZ metric (and the Levenshtein metric) we introduce a new *transformation*, which B applies on y before computing its signature (see Section 4.2). The most interesting property of this transformation is that it converts the LZ metric to Hamming metric; hence it may be helpful in nearest neighbor search problems for sequences [Ind99]. The second round involves the actual exchange of documents via correcting the differences (see Section 5). Here A computes and sends B a deterministic ID for x which is unique among all strings that are in the $\hat{d}(x, y)$-neighborhood of x. By the use of known techniques in graph coloring or error-correcting codes, the size of this ID can be bounded above by $c \log |y| \cdot \hat{d}(x, y)$ bits, where c is a small constant. These protocols work correctly with probability $(1 - \epsilon)$, a user specified confidence parameter. For Hamming distance, the communication complexity is $c' d(x, y) \log |x|$, where c' is a constant determined by ϵ only. For the LZ metric or the Levenshtein metric, the communication cost becomes $c' d(x, y) \log^3 |x|$.

It is possible to save a factor of $\log^2 |x/d|$ in the communication complexity of the above protocols by spreading the distance estimation over $O(\log d(x, y))$ rounds. In a given round r, A again computes and transmits to B a unique ID for x in its 2^r-neighborhood, consisting of 2^r bits. B computes a candidate $\hat{x}$, which is identical to x if $d(x, y) \leq 2^r$, and sends A a fingerprint of $\hat{x}$ for verification.

We also provide simpler protocols that locate the differences between x and y in divide-and-conquer fashion (see Section 6). These protocols make extensive use of fingerprints for searching and comparing substrings of x and y. They require $O(\log |x|)$ rounds and have a communication complexity of $O(d(x, y) \cdot \log^2 |x|)$; their main advantage is that the computations involved are faster.

2 Preliminaries

The Hamming distance between two strings x and y, denoted $h(x, y)$, is of interest when only single characters are altered. This distance is insufficient to capture more general scenarios. If we are comparing an edited document to an earlier version, a better measure is the edit (or Levenshtein [Lev66]) distance, denoted $e(x, y)$. Here, three basic operations, namely inserting, deleting, and replacing single characters, are each assigned unit cost. The distance between two strings is then the minimal number of edit operations required to transform one into the other. Clearly, $e(x, y) \leq h(x, y)$. Edit distance still does not fully capture the appropriate measure of distance for many circumstances; e.g. a paragraph may be moved within a document, or a long sequence of DNA may be copied. Motivated by analogy with Lempel-Ziv data-compression algorithms, we define a new measure between strings below.

DEFINITION 2.1. *The LZ distance between strings x and y, denoted $lz_D(x, y)$ is the minimum number of single characters or substrings of y or of the partially built string which are required to produce x from left to right.*

Example. Suppose x = aabcdefgabc and y = aaaaaaaaaaaaaaa. Then $lz_D(x, y) = 8$ and $lz_D(y, x) = 4$. This can be seen by forming appropriate parsings of the strings: x = (aa)(b)(c)(d)(e)(f)(g)(abc), and y = (aa)(aa)(aaaa)(aaaaaaaa). As $lz_D(x, y) \neq lz_D(y, x)$, the LZ distance does not provide a metric.

We now describe a companion distance measure which is similar to LZ distance but provides a metric.

DEFINITION 2.2. *The LZ metric distance between two strings x and y, denoted $lz_M(x, y)$, is defined as the minimum number of operations of copying a substring, deleting a repeated substring, or inserting, deleting or changing a single character required to transform x into y.*

LEMMA 2.1. *The LZ metric distance provides a metric on finite strings from any constant size alphabet.*

Proof. The lemma follows from the observation that any distance defined by assigning unit cost to a number of reversible operations and counting the shortest sequence of such operations as the distance will necessarily define be metric. □

Intuitively, the LZ metric measures the amount of information that must be exchanged for two parties to know each other's string. If an oracle knows both x and y and wishes to broadcast a single message so that someone with y could calculate x and vice-versa, a good bound on the size of this message is $O(lz_M(x, y) \log(|x| + |y|))$. Exact computation of this metric is hard. This does not deter us; we are interested in considering the success of our protocols in terms of this measure, and approximating distances.

LEMMA 2.2. $lz_M(x, y) \leq lz_D(x, y) + lz_D(y, x)$.

Proof. Consider a particular transformation of x into y which extends x to the string xy by successive 'copy' operations at the right hand end, and then removes x using 'deletes' from right to left through x. This transformation can be performed using no more than $lz_D(x, y)$ copy operations and $lz_D(y, x)$ deletions. □

LEMMA 2.3. $lz_D(x, y) \leq 2e(x, y) + 1$ *and*
$lz_M(x, y) \leq e(x, y)$.

Proof. If $e(x, y) = k$ then y can be parsed into $k + 1$ substrings from x separated by at most k single characters. The first inequality follows. The second is trivial. □

Remark. Lemma 2.3 shows that any protocol that is communication-efficient in terms of LZ metric distance is also communication-efficient in terms of the edit distance. For the purposes of distance estimation we do not treat the protocols for edit distance separately from the protocols for LZ metric distance.

3 Bounds on communication

To examine the performance of the proposed protocols, we need a lower bound on the amount of communication required. The obvious lower bound is the number of bits that would be required for B to compute x given y. If A already knew y in addition to x, A could send a single message containing sufficient information to derive x from y. If, for any y, there are

at most m strings that x could possibly be, then $\lceil \log m \rceil$ bits are necessary. [2] The number of such possible strings is determined by the distance between x and y. We shall denote by k the quantity $|\Xi| - 1$, where Ξ is the alphabet from which our strings are drawn.

LEMMA 3.1. *The amount S of communication needed to correct h Hamming errors between strings x and y ($|x| = |y| = n$) is bounded as: $h \log(nk/h) \leq S \leq \lceil h(\log(nk/h) + 2 \log\log n) \rceil$, provided $h \leq n/2$.*

Proof. The number of strings which satisfy $d(x, y) = h$ given x is precisely $\binom{n}{h} k^h \geq \left(\frac{nk}{h}\right)^h$ (shown by induction in h). We take the logarithm of this quantity to get the lower bound.

For the upper bound, we consider a method to encode the location of each Hamming error, and the correct character to go there. This can be accomplished using $h(\log(kn/h) + \log\log n + 3)$ bits. □

LEMMA 3.2. *The amount S of communication needed to correct e edit operations is bounded as: $e(1 + \log(n(|\Xi - 1|/e)) \leq S \leq \lceil e(1 + \log(|\Xi|(n + 1)/e) + 2 \log\log n) \rceil$, where, n denotes the length of the longer string.*

Proof. The lower bound follows by considering strings generated by only reversals or insertions. The number of such strings which satisfy $e(x, y) = e$ given x is at least $\sum_{i=0}^{e} \binom{n}{i} k^i \binom{n}{e-i} k^{e-i}$. Using the methods of Lemma 3.1 bounds this quantity from below with $(\frac{2kn}{e})^e$. Taking the logarithm of this expression gives the required lower bound.

For the upper bound, we again consider a bitwise encoding of the operations. It is possible to use a bit flag to denote whether each edit is an insertion or a change, and use $\log |\Xi|$ bits to code the character concerned. In the case of a deletion, this is represented as a 'change', but using the code of the existing character at that location. Using a similar location coding as above gives this scheme an overall cost of $e \log(|\Xi|(n + 1)/e)$. □

LEMMA 3.3. *The amount S of communication needed to correct strings such that $lz_M(x, y) = l$ is bounded by $S \leq 3l \log n$*

Proof. We show a bitwise encoding of the lz_M operations to give the upper bound. A 'copy' operation can use $\log n$ bits to specify each of the start, length of substring and destination. Provided n is more than $|\Xi|$ we require no more than $l \log n^3$ bits.[3] □

4 Distance estimation

In this section we show how to estimate the distance between two strings using a sublinear amount of communication.

[2] All logarithms will be taken to base 2

[3] This follows from observing that there is no need for the length of any intermediate string to contain more than one copy of any substring of x or y. The total length of such substrings is $O(n^3)$.

4.1 Hamming metric. In this section we show how one of the communicating parties, say B, can compute an estimate of the Hamming distance between its binary string y and the string x of the other party, A. At the core of this process is a simple hash function which is more likely to collide for strings that are closer together; it is also used for different purposes in [AMRT96] and [KOR98]. Estimating Hamming distance in the context of communication complexity has not been addressed before, although the closely related problem of approximate nearest neighbor search in Hamming space has been studied independently in [IM98] and [KOR98].

Denote by n the length of x (and y), and denote by $\hat{h}(x,y)$ the estimate of $h(x,y)$ computed by B. We will need only a single round of communication during which B receives an $O(\log n)$-bit signature of x from A. In Section 5 we describe how this distance estimate is used by B to compute and send A a unique ID for y, which can be decoded by A. This ID will consist of $O(\hat{h}(x,y)\log n)$ bits hence it is important not to over-estimate $h(x,y)$.

The protocol is randomized and can result in an under-estimation of $h(x,y)$ with probability no more than a user-specified confidence parameter ϵ. Because an under-estimation will result in A computing y erroneously, we would like to choose ϵ to be small. We show that the probability of over-estimating $h(x,y)$ by more than a constant factor (which we will specify later) is small as well.

The signature of x is computed in $O(\log n)$ iterations as follows. We assume that two identical random number generators are accessible to both parties A and B. In the i^{th} iteration, A uses its random number generator to uniformly and independently pick a sequence of β^i integers from the range $\{1,\ldots,n\}$, denoted by $r_1,\ldots,r_{\beta^i}$ for some $\beta > 1$ which will be specified later. This sequence (which is available to B as well) is used to obtain the sample $\hat{x}_i = x[r_1],\ldots,x[r_{\beta^i}]$ from x, which is then hashed to $m_i(x) = \bigoplus_{j=1,\ldots,\beta^i} x[r_j]$ as per Lemma 13 of [AMRT96]. (This is similar to the β-test in [KOR98]. The difference is that each bit of the signature is used as a binary test on estimating the Hamming distance rather than as a test to decide which string among two strings A and B is closer to a query string.) The message of A will be $m(x) = m_1(x),\ldots,m_{\log_\beta n}(x)$ and hence will include a total of $\log_\beta n$ bits.

At the same time B computes $m(y) = m_1(y),\ldots,m_{\log_\beta n}(y)$ by using the same procedure. After receiving $m(x)$, B computes $m(x)\oplus m(y) = m_1(x)\oplus m_1(y),\ldots,m_{\log_\beta n}(x)\oplus m_{\log_\beta n}(y)$. Then, B uses the smallest i for which $m_i(x)\oplus m_i(y) = 1$ to compute an estimate $\hat{h}(x,y)$ for $h(x,y)$. Below we describe how B computes its estimate based on the distribution of $m(x)\oplus m(y)$.

Observe that $m_i(x)\oplus m_i(y) = \bigoplus_{j=1,\ldots,\beta^i} x[r_j]\oplus y[r_j]$. Therefore $\Pr[m_i(x)\oplus m_i(y)=1]$ is equal to the probability of getting an odd parity out of β^i random i.i.d. bits, each of which is 1 with probability $p = h(x,y)/n$. This can easily be computed by the following well known lemma, a short proof of which is given in the appendix.

LEMMA 4.1. *Let $b_1,\ldots,b_k$ be independent Boolean random variables with expectation p. $\Pr[\sum b_i$ is even $] = (1+(1-2p)^k)/2$.*

Let $C(k,p) = (1+(1-2p)^k)/2$. $C(k,p)$ can be tightly bounded above and below as follows (see appendix for a proof).

LEMMA 4.2. *For all $k \geq 2$, and $0 \leq p \leq 1$:*
(i) $e^{-kp} \leq C(k,p) \leq (1+e^{-2kp})/2$;
(ii) if $e^{-\frac{2}{3}kp} \geq \frac{\sqrt{5}-1}{2}$, i.e. $kp \leq 0.7218\ldots$, then $C(k,p) \leq e^{-\frac{2}{3}kp}$.

Consider the process of taking the parity of geometrically increasing sequences of random samples of bits. The ratio between successive sample sizes is $\beta > 1$. Suppose we have such a sequence increasing up to size k. Define $Z(k,p,\beta)$, the probability that all parities are zero, to be $\prod_{i\geq 0}^{u} C(k/\beta^i, p)$, where $u = \log_\beta k$.

LEMMA 4.3. *$Z(k,p,\beta) \geq e^{-kp/(1-1/\beta)}$;*
and $Z(k,p,\beta) \leq e^{-\frac{2}{3}kp(1-1/\beta)}$ provided $e^{-\frac{2}{3}kp} \geq \frac{\sqrt{5}-1}{2}$.

The proof is immediate from the definition of $Z(k,p,\beta)$ by using the preceding lemma.

The nature of our protocol is such that it *fails* if B *underestimates* p, i.e., if this first sample of odd parity (at k) occurs later than we would expect. For any $\epsilon > 0$, B will make a cautious estimate $\hat{p}$ ($= \hat{h}(x,y)/n$) by choosing $\hat{p}$ such that $Z(k/\beta, \hat{p}, \beta) \leq \epsilon$, since the probability of not getting odd parity before k is at most ϵ if $p \geq \hat{p}$. We will satisfy $Z(k/\beta, \hat{p}, \beta) \leq \epsilon$ by taking $e^{\frac{-2k\hat{p}}{3(\beta-1)}} = \epsilon$, i.e. $\hat{p} = \frac{3(\beta-1)\ln 1/\epsilon}{2k}$, and ensuring $e^{\frac{-2k\hat{p}}{3}} \geq \frac{\sqrt{5}-1}{2}$. The latter holds if we choose β so that $1 < \beta \leq 1 + \frac{1}{\ln 1/\epsilon}\ln(\frac{\sqrt{5}-1}{2})$.

As a result, the protocol estimates $h(x,y)$ to be $\hat{h}(x,y) = \hat{p}\cdot n = n \cdot \frac{3(\beta-1)\ln 1/\epsilon}{2k}$, and ensures that the probability of an underestimation is at most ϵ. The proof of the following lemma is given in the appendix.

LEMMA 4.4. *In the protocol $\Pr[\hat{p} \geq \alpha p] \leq 1/2$, i.e., the probability of overestimation of p by a factor of α is bounded by a constant, for $\alpha \geq \frac{3\beta \ln 1/\epsilon}{2\ln 2} \geq 1$.*

The communication complexity of the distance estimation protocol is $O(\log_\beta n)$, which is $O(\log(1/\epsilon)\log n)$, and involves a single round of communication. By increasing the number of rounds to $\log_\beta \hat{h}(x,y)$, the communication complexity of this protocol can be reduced to $\log_\beta \hat{h}(x,y)$ which is bounded above by $\log_\beta(\alpha h(x,y))$ with probability greater than $1/2$.

4.2 Converting "limited" LZ distance to Hamming distance. Our protocol for estimating $lz_M(x,y)$ is based on a transformation we introduce, which converts the LZ metric to Hamming metric. Before we give the details of the transformation, we first focus on a simpler metric that we call the limited LZ metric and provide a transformation to convert the limited LZ distance to Hamming distance. This will provide the intuition behind the more general transformation which we describe later.

DEFINITION 4.1. *Let x and y be two binary strings of length $n = 2^k$. Define the* limited LZ distance *between x and y, denoted $ltd(x,y)$, as the minimum number of the following "edit" operations applied on x to obtain y: (1) changing any given bit of x, (2) swapping any two non-overlapping substrings in the form $x[i2^l + m : i2^l + p]$ and $x[j2^l + m : j2^l + p]$ where $0 < m < p < 2^l$ (we call these substrings aligned), and (3) replacing any substring of the form $x[i2^l + m : i2^l + p]$ which has an identical non-overlapping copy in x, by a copy of*

any non-overlapping substring of the form $x[j2^l + m : j2^k + p]$ *(again,* $0 < m < p < 2^l$*). Notice that if we apply on* y *the reversal of the operations applied on* x*, we obtain* x *back; so* $ltd(x, y) = ltd(y, x)$*, and hence* $ltd(\cdot, \cdot)$ *is a metric.*

Given a binary string z of size $n = 2^k$, let the 2^i-binary histogram of z, denoted $LT_i(z)$, be the binary vector of size 2^{2^i}, whose j^{th} dimension, denoted $LT_i(z)[j]$, is 1 if there is at least one substring $z[l2^i + 1 : (l + 1)2^i]$ (for $l = 0, \ldots, n/2^i - 1$) which is equal to the binary representation of j; otherwise $LT_i(z)[j] = 0$.

Example. Let $z = 111011$; to compute $LT_1(z)$ we consider only the substrings $z[1 : 2] = 11$, $z[3 : 4] = 10$ and $z[5 : 6] = 11$ and deduce that $LT_1(z)[00] = 0,\ LT_1(z)[01] = 0,\ lT_1(z)[10] = 1,\ LT_1(z)[11] = 1$ and hence $LT_1(z) = (0, 0, 1, 1)$.

Definition 4.2. *Let* $LT(z)$*, the* limited LZ transformation *of* z*, be the concatenation of all* $LT_j(z)$ *for* $j = 0, \ldots, k$*.*

Definition 4.3. *We define the* limited parse tree *of a string* z *of size* 2^k *as the complete binary tree of depth* k *whose nodes are labeled with substrings of* z *as follows. The root of the tree is labeled with* z *itself. The labels of the children of any given internal node with label* w *are* $w[1 : |w|/2]$ *and* $w[|w|/2 + 1 : |w|]$*.*

Theorem 4.1. *If* x*,* y *are strings of length* 2^k*,* $\frac{1}{2}ltd(x, y) \leq h(LT(x), LT(y)) < 8k \cdot ltd(x, y)$

Proof. (i) $h(LT(x), LT(y)) < 8k \cdot ltd(x, y)$: We show that each permitted operation of the distance affects fewer than $8k$ components of $h(LT(x), LT(y))$. (1) Changing one bit of x could change the value of at most two components of $LT_i(x)$ at each level, in total $2k$. (2) Consider a swap of two substrings length $L > 1$, and let $l = \lfloor \log L \rfloor$. In the limited parse tree, we count the nodes which cover part of a substring to be swapped and part of the string which remains static. Each such node corresponds to the change of at most two components of $LT(x)$ (since one component can go from one to zero as another goes from zero to one). The number of such nodes is bounded above by $2(k + l) - 1$ (count k from the left side and l from the right side of both substrings). Since $l \leq k - 1$, each swap affects fewer than $8k$ components of $LT(x)$. (3) Consider an 'overcopy' of a substring length $L > 1$, and let $l = \lfloor \log L \rfloor$. The argument proceeds as for case (2), except we observe that only the string which is copied onto changes, while the string which is copied remains the same, hence the number of affected nodes is halved, giving a bound of $4k$ altered substrings.

(ii) $\frac{1}{2}ltd(x, y) \leq h(LT(x), LT(y))$: We describe a procedure that obtains y by applying at most $2h(LT(x), LT(y))$ operations on x.

A preprocessing step performs a left-to-right breadth first search on the limited parse tree of x, to "protect" certain substrings. We protect a substring of x which appears in the limited parse tree if that substring occurs in y and it is the first occurrence of this substring in x. The aim of this step is to ensure that substrings in the target string y which exist in x are not damaged in the process of transforming x. Protected strings may not have any of their substrings copied over, or swapped, nor may any bit be changed. However,

any substring may be copied since this does not 'damage' the protected string. Protecting substrings does not involve any operations, and so has no cost.

The procedure to transform x into y then proceeds as follows. We consider manipulating substrings of length 2^i for $0 \leq i \leq k$. We will only consider substrings in the limited parse trees of x and y. Our inductive hypothesis is that at the end of manipulation of length 2^i substrings, then the set of these substrings present in the modified string will be a superset of those of the same length present in y. Clearly if we can maintain this hypothesis up to substrings length 2^k then we will have succesfully transformed x into y since there can be only one sub-string of this length (the string itself).

We will show that the Hamming distance of the limited histograms gives a bound on the number of operations needed by allocating "credits" to entries in the histrograms which will pay for the operations. For each substring z such that $LT(x)[z] = 0$ and $LT(y)[z] = 1$ we allocate two credits. Similarly, we also give two credits to each substring z for which $LT(x)[z] = 1$ and $LT(y)[z] = 0$. The number of credits allocated is exactly $2h(LT(x), LT(y))$. Of these two credits, at most two will be spent at any one time. Credits from a substring of x will be spent on removing the last occurrence of that substring from the modified string; credits for a substring of y will be spent to create that substring. Since we create only one copy of each substring, and do not recreate uneeded copies of substrings of x, each pair of credits will be used at most once. Not all credits will necessarily be used.

We shall now proceed to spend these credits. The base case is to ensure that the set of length 1 substrings in x is a supersetof those present in y. If x consists of all 0s or all 1s, then we shall change the first unprotected bit, which is paid for by a credit from that bit in y. Otherwise, no action is necessary. From this point onwards, changing a single bit is the same as overcopying a single bit, and so we consider that there are only two feasible operations – swapping and overcopying substrings.

The inductive case is that there exists in the modified string, x_i, at least one copy of all length 2^i substrings of y. We proceed to modify x_i to form x_{i+1}. When we consider building substrings of length 2^{i+1}, strings of length 2^i and shorter lose their protection. We pick an arbitrary length 2^{i+1} substring y' of y that is not represented in x_i and try to build it in the first slot from the left whose substring is not protected. Because y' did not exist in x (else it would have been protected and hence preserved up to this point), we use the two credits for this substring to build it. There is a copy of the left and right half of y' in x_i by our inductive hypothesis, so we spend one credit on each half bringing the two together. We now perform a case split on what is in the intended place of the left (or right) half, and show how this can be dealt with using a single credit.

(i) A substring which appears in x but not in y which has more than one copy in x_i: we can overcopy the appropriate half of y' in using a single move.

(ii) A substring which appears in x but not in y and is the only occurrence of this substring in x_i: we use the two credits that this substring was allocated to overcopy the left and right halves of the half of y'. This process may proceed recursively if one or both halves also represent substrings of x which have only one occurrence in x_i, in which case credits for these substrings may be used as well.

(iii) A substring which appears in y and is not the only copy of this substring in x_i: We can copy over this substring without problem since there exist other copies which can be used

later if necessary, and it is not protected (by our precondition in the above paragraph).
(iv) A substring which appears in y and is the only copy of this substring in x_i: This case requires the most care. If the source substring (the substring we wish to place) is not part of a protected string, then we can swap this substring instead of over-copying it, at unit cost. If the source substring is part of a protected string, then we need to change our plan: depending on the exact situation we can instead build a substring which uses the source substring, or else build y' in a different place.
(v) If it happens that the two halves are in the same slot, but in the wrong order, then we can swap them into place with a single move, which is charged to y'.

Because of the way we have formed x_i, no other cases are possible. This procedure is repeated until we have built a representative of each length 2^{i+1} substring of y that was not present in x_i, and this new string is our x_{i+1}. The procedure concludes when we have built x_k, which is y. □

Remark. The limited LZ transformation of a binary string of size n is an $O(2^n)$ dimensional binary vector which is very sparse – only $O(n/\log n)$ of the entries are non-zero. To decrease the size of the signature, we start with a very large sample (of size $O(2^n/poly(n))$ and again increase the sample size by a factor of β in subsequent iterations. This will not affect the communication complexity of the protocols or the confidence bounds provided. We cannot bias our sample towards non-zero components, as suggested in a different context in [IM98], since the sampling procedure must be completely replicable by the other party.

4.3 Converting LZ distance to Hamming distance. The procedure for converting the LZ distance to Hamming distance is based on a generalization of the limited LZ transformation by removing the restriction that only the aligned substrings can be swapped. The main tool which enables us to obtain this (general) LZ transformation is the locally consistent parsing technique, LCP [ṢV96].

Given a string z of size n, its LZ transformation includes the binary histograms of the *level-i cores* of z for all $i = 0, \ldots, \log n$, computed by LCP. [4] The cores are special substrings that are determined by the size of the alphabet which satisfy the following important property: The number of level-i cores of a given string z is upper bounded by $n/2^i$ and is lower bounded by $n/3^i - 2(\log^* n + 3)$ [ṢV96]. This enables us to define the level-i-binary histogram of z, denoted $T_i(z)$, as the binary vector of size $O((3\log^* n)^i)$, whose j^{th} dimension, denoted $T_i(z)[j]$, is defined to be the number of *all* level-i-cores that are equal to the binary representation of j. Now we define the *LZ transformation of* z as the concatenation of all $T_j(z)$ for $j = 0, \ldots, \log n$, and denote it by $T(z)$. Note that we do not have the restriction of alignment between strings, hence we can accommodate not only aligned swaps but also unaligned swaps, insertions and deletions. This leads us to the following theorem whose proof is left to the full paper.

THEOREM 4.2. *For constants c' and c'',*
$c' \cdot lz_M(x, y) \leq h(T(x), T(y)) \leq c'' \cdot \log^2 n \cdot lz_M(x, y)$.

[4] Compare this to the limited LZ transformation which comprises of the binary histograms of length 2^i substrings of the input, for $i = 0, \ldots, \log n$.

5 Correcting differences

Let $d(x, y)$ be a metric over a set X defined by the transitive closure of a symmetric relation R which is assigned unit cost – that is, $d(x, y)$ is the minimum n such that $R^n(x, y)$. Suppose that there are two individuals, A who holds $x \in X$ and B who holds $y \in X$, such that $d(x, y) \leq l$ for some known l. The range of d is the natural numbers, and we will refer to this as the distance. Let G_l be the undirected graph whose vertices are X and whose edges are $\{(x, y) : d(x, y) \leq l\}$.

LEMMA 5.1. $\deg(G_{2l}) \leq (\deg(G_l))^2$.

Proof. $\deg(G_{2l})$ corresponds to the greatest number of vertices at a distance at most $2l$ (in the graph corresponding to R) from any particular vertex, x. As the metric is defined by unit cost operations, these vertices will be those which are at most l from all vertices at most l from x. Since from each vertex there are at most $\deg(G_l)$ vertices at distance at most l, the lemma follows. □

Following the model of Orlitsky, we shall employ a hypergraph H whose vertices are the members of X and whose hyperedges are $e_y = \{x : d(x, y) \leq l\}$. Both parties can calculate this hypergraph independently and without communication (if some parameter of the graph is not implicit, A can prepend this information to the message with small additive cost). They can then use a deterministic coloring scheme to color the hypergraph with $\chi(H)$ colors. A's message is the color of x. By the coloring scheme and the specification of the problem, B knows that x must be amongst the vertices in e_y, and can use the color sent by A to identify exactly which it is. We can place upper and lower bounds on the single message complexity by considering this scheme. Since we must be able to distinguish x among the vertices in the hyperedge x, at least $\log(\deg(H))$ bits must be sent overall. To send the color of x, we need $\lceil \log \chi(H) \rceil$ bits [KN97]. Let S be the number of bits necessary in a single message to allow B to calculate x (the single message cost). So $\log(\deg(H)) \leq S \leq \lceil \log(\chi(H)) \rceil$.

LEMMA 5.2. $\chi(H) \leq \deg(G_{2l})$.

Proof. $\chi(H)$ is the number of colors required to color the vertices of H such that any two vertices which are both at most distance l from any point x are colored differently. Consider y and y' such that $d(x, y) \leq l, d(x, y') \leq l$. Since d is a metric, $d(y, y') \leq d(y, x) + d(x, y') = d(x, y) + d(x, y') \leq 2l$.

Any coloring which ensures that any two points within distance $2l$ have different colors is therefore a coloring of the hypergraph H. A coloring of G_{2l} achieves this, showing that $\chi(H) \leq \chi(G_{2l})$.

On the assumption that G_{2l} is not complete, is connected and has degree of three or more (which will be satisfied by the examples in which we are interested), then $\chi(G_{2l}) \leq \deg(G_{2l})$, and so the lemma follows. □

THEOREM 5.1. $\log(\deg(G_l)) \leq S \leq 2 \log(\deg(G_l))$.

Proof. It is clear that $\deg(G_l) = \deg(H)$; in both cases the degree of each vertex corresponds to the number of vertices a distance at most l away. We combine this with Lemma 5.2 and Lemma 5.1 into the fact that $\log(\deg(H)) \leq S \leq \lceil \log(\chi(H)) \rceil$ to prove the theorem. □

Thus, under certain conditions, a single message can be at worst twice the lower bound on its length for any number of messages. A similar theoretical result to that shown here is given by Orlitsky [Orl91]. By considering a more restricted class of functions, we can construct a concrete protocol which achieves the above bound. Note that our upper bound is related to the degree of a graph which we can calculate, and algorithms exist which will color a graph with this number of colors; hence it is achievable in practice. The constraints placed upon the metric d may seem restrictive, so we show how our earlier distances (Section 2) fit the restrictions.

DEFINITION 5.1. *A protocol between two communicating parties A and B is* non-trivial *if it enables A to transmit a string x to B, under a stated restriction on the distance between x and y, using fewer than $|x| \log |\Xi|$ bits of communication in total.*

Hamming distance is a metric on the space of strings of length n over an alphabet Ξ, and is induced by defining a symmetric relation on strings which differ in a single character. For $l < n/2$, G_{2l} is not complete, and for any non-trivial n, G_{2l} will have degree higher than two: $\deg(G_l) = \sum_{i=1}^{l} \binom{n}{i} (|\Xi| - 1)^i$. There is no simple closed form for this expression, so instead we consider the upper bound on this quantity for up to l Hamming differences described in Lemma 3.1. This uses $l \log((|\Xi| - 1)(n/l)) + o(l \log \log(n/l))$ bits, hence we have the result that up to l Hamming errors can be corrected using a single message with asymptotic cost at most $2l \log((|\Xi| - 1)(n/l))$.

Edit distance is defined by charging unit cost to insertions, deletions or alterations of a character. Again, for $l < n/2$ (n denotes the length of the longer string), G_{2l} is not complete and will have degree more than two. Also, there is no concise exact expression for the degree of G_l, but we can use the binary encoding (Lemma 3.2) for an upper bound on $\log(\deg(G_l))$. We can encode up to e edit operations using at most $e(\log 2(|\Xi|(n+1)/e)) + o(e \log \log n)$ bits, so asymptotically $2 \log \deg(G_l) \leq 2l \log(|\Xi|(n+1)/l)$ which is non-trivial for $l < n/2$.

LZ metric distance also conforms to the requirements. Using the simple encoding from Lemma 3.3 we can achieve a single message cost of $18l \log n$ bits, non-trivial for $l \leq \frac{n \log |\Xi|}{18 \log n}$.

5.1 Multi-round protocols. It is possible to have multiple round adaptations of these protocols. In the first round the users run the protocol with the assumption that $d(x, y)$ is upper bounded by some small constant. They then use a hash function over their strings to see if they agree. If not, they double the distance, and repeat the protocol, until they agree. This requires at most $\log n$ rounds but gives a more precise estimate of $d(x, y)$. For the LZ metric and the Levenshtein metric this amounts to a saving of a $\log^2 |x|$ factor in communication complexity.

REMARK 5.1. *While these protocols are communication-efficient, they are computationally expensive. For the Hamming distance, an efficient coloring can be achieved by Reed-Solomon codes (see [AGE94]). Since any one-round protocol corresponds to an error-correcting code, an efficient coloring scheme for edit distance or the LZ metric would lead to error-correcting codes for these systems. We have shown concrete methods to construct such codes; it is open whether these codes could be calculated more efficiently.*

6 Computationally efficient protocols

We first consider the following abstracted problem: given a string of length n, locate h distinguished characters by using queries of the form whether such a character exists in any specified substring. This can be solved by dividing the string into two equal size substrings and querying them. If a substring is free of distinguished characters, then we do not consider it further. Otherwise it is further divided into two each of which is queried inductively.

LEMMA 6.1. *An upper bound on the number of queries made in this protocol is* $2h \log(2n/h)$.

Proof. (sketch) The worst case is when the distinguished characters are distributed evenly throughout the string. If we consider descending the limited parse tree of the string (Definition 4.3), we must query all $2h-2$ substrings in the first $\log h$ levels, and $2h$ substrings in each of the remaining $\log n - \log h$ levels. □

Each query gives us one bit of information, so here we are gaining no more than $2h \log(n/h) + O(h)$ bits of information. This is asymptotically only twice the minimum (this follows from Lemma 3.1). We will now apply this abstract problem to our various distance measures.

6.1 Hamming distance. The problem of locating the Hamming errors, given an upper bound on the number of errors, is similar to the problem of group testing (also observed by Madej [Mad89]). We wish to group samples and perform a test which will return either 'all the same' or 'at least one mismatch'. The differences are that we have an ordering of the samples, given by their location in the string, and that we have to contend with the problem of false negatives We use negative to mean that the test indicates no errors; false negatives are an inevitable consequence of using fewer than n bits of communication. Our tests will be based on fingerprinting; i.e., if a substring in one string hashes to the same value as the corresponding substring in the other, then we take this as evidence that the two substrings match. Note that there is no danger of false positives; if a test leads us to believe that the substrings are different, then there is zero probability that they actually match.

The following scheme is a straightforward way to locate and correct differences, and is similar to that proposed in [Met83]. Mapping our terminology onto that of the above problem, the locations of Hamming differences between the two strings are the distinguished characters. We shall use the proposed simple divide and conquer algorithm to locate them. The test we perform involves communication: in each round, one party will send the value of a hash function for the substrings in question, and the other will calculate the value for the corresponding substrings of their string. If they agree, then there is (with high probability) no difference between the two substrings; otherwise there is (with certainty) some discrepancy between them. The reply to the message is a bitmap indicating the success or failure of each test. We must choose our functions so that the overall probability of success of the procedure is high.

If we use linear hashes relative to a randomly chosen base B, then the probability of two arbitrarily chosen sequences having the same hash value is $1/B$. If we treat each test as independent, then we want, say, $(1 - 1/B)^{2h(\log 2n/h)} \geq p_c$.

LEMMA 6.2. *For* $B > 1$, $1/B < -\ln(1 - 1/B) < 1/(B - 1)$.

Proof. For the second inequality, $-\ln(1-\frac{1}{B}) = \ln(1+\frac{1}{B-1}) < \frac{1}{B-1}$. □

LEMMA 6.3. *With an estimate of the Hamming distance, $\hat{h}$, this protocol succeeds with probability p_c and communicates no more than $2h(\log 2n/h)\log\left(\frac{2\hat{h}\log(2n/\hat{h})}{\ln 1/p_c}\right)$ bits.*

Proof. Follows immediately by using Lemma 6.2 to choose B to be at least $\lceil\frac{2h(\log 2n/h)}{\ln 1/p_c}\rceil$ and combining this with Lemma 6.1. □

This protocol is *non-trivial* for $h = O(n/\log n \log\log n)$. The overall cost is a factor of $O(\log \hat{n} \log n)$ above the optimal; however, this cost comes from the size of the hash values sent. With regard to the number of hash values sent, we have shown it is about twice the optimal of any possible scheme which uses hash functions to give a binary answer to a question. The number of rounds is logarithmic in the length of the string, as we can progress one level of the limited parse tree in each round.

6.2 Edit Distance. We next translate the above approach to deal with the edit distance. A similar scheme is proposed but not analyzed in [SBB90].

It is not immediately apparent how to map the edit distance problem onto the abstract problem presented above. However, observe that the way that differences were located was by a process of elimination: identical substrings were found, until all that remained were disparate substrings. We may use the same kind of hierarchical approach to identify common fragments. It is no longer the case that substrings are aligned between the strings; however, we know that matching substrings will be offset by at most the edit distance.

The party B receiving the hash values must therefore do more work to identify them with a substring. B has a bound $\hat{d}$ on the edit distance; if a substrings is unaltered by the editing process, then it will be found at a displacement of no more than $\hat{d}$ from its location in A's string. So B calculates hashes of substrings of the appropriate length at all such displacements left and right from the corresponding position in its string, testing each one to see if it agrees with the sent hash. If they do agree, it is assumed that they match, and so B now knows the substring at this location in A.

We consider an optimal edit sequence from x to y and how it affects the limited parse tree representing the splitting of x. Deletions and replacements affect single characters at leaf nodes. An insertion of any number of consecutive characters between two adjacent characters is considered to occur at the internal node which is the lowest common ancestor of the pair. The protocol must traverse this tree, computing hashes on every node whose subtree contains any of these edit operations.

LEMMA 6.4. *This protocol succeeds with probability p_c and communicates no more than $4d\log(2n/d)\log(2\hat{d}) + O(d\log n \log\frac{\log n}{\ln 1/p_c}))$ bits.*

Proof. Observe that the worst case is exactly as before – if (almost) all errors are deletions or alterations (since we have to descend further down the tree to discover these). Certainly, in the worst case, the number of hashes sent will be that given by Lemma 6.1, $2d\log(2n/d)$. We must compare each hash sent with up to $2\hat{d}+1$ others in the worst case. As

before, we want the probability of an error to be no more than a constant, which gives $\left(\frac{B-1}{B}\right)^{2\hat{d}(2\hat{d}+1)\log(2n/\hat{d})} \geq p_c$. We again use Lemma 6.2 to choose B, and the lemma follows. □

This expression is $O(d\log(n/d)\log\hat{d})$, asymptotically $O(\log\hat{d})$ above the optimal.

6.3 LZ distance. Consider a variation of the above abstract problem. Instead of h distinguished characters, suppose that there are l "dividers" which are positioned between characters, such that there is at most one divider between any two adjacent characters. This problem is reducible to the distinguished character problem: there are $n-1$ locations where dividers can be placed. If our queries are whether there are any dividers within a substring, then clearly the same protocol will suffice, with the same number of hashes.

Rather than descending the tree in parallel, the protocol performs a depth-first search. We consider a notional optimal parsing of a string x relative to a string y with $lz_D(x,y) = l$. The protocol proceeds as follows: B maintains a dictionary of 'resolved' hashes, containing the mapping from hash values to each substring of length 2^i. This is initialized with all possible hashes over y, and is augmented with information about x as it is learned. [5] On receiving a pair of hashes, B tries to use this dictionary to identify them. B indicates to A how far along x he can identify. A, who maintains a record of the progress of B, sends two hash values on strings made by splitting the first remaining unresolved substring into two equal pieces. The number of rounds involved is at most $l\log(n/l)$.

LEMMA 6.5. *This protocol uses no more than* $4l\log(2n/l)\log(\frac{2n}{\ln 1/p_c})$ *bits of communication and succeeds with probability* p_c.

Proof. In an optimal parsing, x is parsed into l pieces, each piece of which is a substring present in y or earlier in x, or a single character. The number of hashes necessary for B who holds y to identify x is that needed to locate the $l-1$ "dividers" which separate the substrings, given by Lemma 6.1. This is $2(l-1)\log(2\frac{n-1}{l-1})$, which is less than $2l\log(2n/l)$.

We are performing many more comparisons between hash values than before, so we require a larger base over which to compute hashes in order to ensure that the chance of a hash collision is still only constant for the whole process. We only compare hashes for substrings of the same length. If we have two strings each of length n then there are fewer than $2n$ substrings of any given length. So there are fewer than $2n^2$ possible pairwise comparisons. Invoking Lemma 6.2 again leads us to choose $B = \lceil\frac{2n^2}{\ln 1/p_c}\rceil$. The lemma then follows. □

If, instead of the full LZ metric, we only imagine the unknown string to be parsed into substrings of the other string and single characters, then it is straightforward to show that performing the divide and conquer algorithm breadth first is $O(\log n)$ above optimal cost under this measure and only uses $\log n$ rounds. In the case where we expect the files to share many substrings, then this approach would be appropriate.

[5] So if B identifies a sub-string of length 16, then not only is the mapping from its hash to the substring added, but so are the 9 substrings of length 8, the 13 substrings of length 4 and the 15 of length 2.

Acknowledgments. We would like to thank the anonymous referees for their careful work and many useful comments. The third author thanks Funda Ergun for all the valuable discussions and her help in writing this paper.

References

[AGE94] Khaled A. S. Abdel-Ghaffar and Amr El Abbadi. An optimal strategy for comparing file copies. *IEEE Transactions on Parallel and Distributed Systems*, 5(1):87–93, January 1994.

[AMRT96] Arne Andersson, Peter Bro Miltersen, Søren Riis, and Mikkel Thorup. Static dictionaries on AC^0 RAMs: Query time $\Theta(\sqrt{(}\log n/\log\log n))$ is necessary and sufficient. In *37th Annual Symposium on Foundations of Computer Science*, pages 441–450. IEEE, 1996.

[BL91] Daniel Barbara and Richard J. Lipton. A class of randomized strategies for low-cost comparison of file copies. *IEEE Transactions on Parallel and Distributed Systems*, 2(2):160–170, April 1991.

[IM98] Piotr Indyk and Rajeev Motwani. Approximate nearest neighbors: Towards removing the curse of dimensionality. In *Proceedings of the 30th Annual ACM Symposium on Theory of Computing (STOC-98)*, pages 604–613, 1998.

[Ind99] Piotr Indyk. Personal communication, 1999.

[KN97] Eyal Kushilevitz and Noam Nisan. *Communication Complexity.* Cambridge University Press, 1997.

[KOR98] E. Kushilevitz, R. Ostrovsky, and Y. Rabani. Efficient search for approximate nearest neighbor in high dimensional spaces. In *Proceedings of the 30th Annual ACM Symposium on Theory of Computing (STOC-98)*, pages 614–623, 1998.

[Lev66] V. I. Levenshtein. Binary codes capable of correcting deletions, insertions and reversals. *Soviet Physics Doklady.*, 10(8):707–710, February 1966. Doklady Akademii Nauk SSSR, V163 No4 845-848 1965.

[Mad89] T. Madej. An application of group testing to the file comparison problem. In *9th International Conference on Distributed Computing Systems*, pages 237–245. IEEE, June 1989.

[Met83] J. J. Metzner. A parity structure for large remotely located replicated data files. *IEEE Transactions on Computers*, 32(8):727–730, August 1983.

[Met91] J. J. Metzner. Efficient replicated remote file comparison. *IEEE Transactions on Computers*, 40(5):651–659, May 1991.

[Orl91] A. Orlitsky. Interactive communication: Balanced distributions, correlated files, and average-case complexity. In *Proceedings of the 32nd Annual Symposium on Foundations of Computer Science*, pages 228–238. IEEE, October 1991.

[PG86] King F. Pang and Abbas El Gamal. Communication complexity of computing the Hamming distance. *SIAM Journal on Computing*, 15(4):932–947, 1986.

[SBB90] T. Schwarz, R. W. Bowdidge, and W. A. Burkhard. Low-cost comparisons of file copies. In *10th International Conference on Distributed Computing Systems*, pages 196–202. IEEE, 1990.

[ŞV96] Süleyman Cenk Ṣahinalp and Uzi Vishkin. Efficient approximate and dynamic matching of patterns using a labeling paradigm (extended abstract). In *37th Annual Symposium on Foundations of Computer Science*, pages 320–328. IEEE, 1996.

Appendix.

Proof for lemma 4.1.
Let $e_j = \Pr[\sum b_i = j] = \binom{k}{j}(1-p)^{k-j}p^j$. The generating function for e_j is $E(x) = \sum e_j x^j = (1-p+px)^k$, so $e_0 + e_2 + \ldots = (E(1) + E(-1))/2 = (1 + (1-2p)^k)/2$. □

Proof for lemma 4.2.
(i) Since $e^{-kp} = C(k,p) = 1$ for $p = 0$, we can show $e^{-kp} \leq C(k,p)$ by proving $\frac{d}{dp}e^{-kp} \leq \frac{d}{dp}C(k,p)$, i.e., $e^{-kp} \geq (1-2p)^{k-1}$, for $k \geq 2$, or $kp \leq -(k-1)\log(1-2p)$. However, $-(k-1)\log(1-2p) = (k-1)(2p + \frac{(2p)^2}{2} + \ldots) \geq (k-1)2p \geq kp$. For the right-hand inequality, $(1-2p)^k \leq e^{-2pk}$ since $1 - 2p \leq e^{-2p}$.

(ii) We want to show that $(1 + e^{-2kp})/2 \leq e^{-\frac{2}{3}kp}$ under the given inequality. Putting $y = e^{-\frac{2}{3}kp}$, we want the inequality $1 + y^3 - 2y = (1-y)(1-y-y^2) \leq 0$ when $1 \geq y \geq \frac{\sqrt{5}-1}{2}$. This is easily checked. □

Proof for lemma 4.4.
The protocol's estimate for p is $\hat{p} = \frac{3(\beta-1)\ln(1/\epsilon)}{2k}$. Let $k' = \frac{3(\beta-1)\ln(1/\epsilon)}{2p}$.

$$\Pr[\hat{p} \geq \alpha p] = \Pr[k \leq \frac{k'}{\alpha}] = 1 - Z(k'/(1+\alpha), p, \beta)$$

$$\leq 1 - \exp\left(\frac{k'p}{\alpha(1-1/\beta)}\right) = 1 - \exp\left(\frac{-3\beta\ln(1/\epsilon)}{2\alpha}\right)$$

If $\alpha \geq \frac{3\beta\ln(1/\epsilon)}{2\ln 2}$, then $\Pr[\hat{p} \geq \alpha p] \leq \frac{1}{2}$. □