Protocols for Asymmetric Communication Channels

Micah Adler[1]
Department of Computer Science
University of Massachusetts
Amherst, MA 01003-4610
micah@cs.umass.edu

Bruce M. Maggs[2]
School of Computer Science
Carnegie Mellon University
5000 Forbes Avenue
Pittsburgh, PA 15213
bmm@cs.cmu.edu

July 3, 2000

Abstract

In this paper we examine the problem of sending an n-bit data item from a client to a server across an asymmetric communication channel. We demonstrate that there are scenarios in which a high-speed link from the server to the client can be used to greatly reduce the number of bits sent from the client to the server across a slower link. In particular, we assume that the data item is drawn from a probability distribution D that is known to the server but not to the client. We present several protocols in which the expected number of bits transmitted by the server and client are $O(n)$ and $O(H(D)+1)$, respectively, where $H(D)$ is the binary entropy of D (and can range from 0 to n). These protocols are within a small constant factor of optimal in terms of the number of bits sent by the client. The expected number of rounds of communication between the server and client in the simplest of our protocols is $O(H(D))$. We also give a protocol for which the expected number of rounds is only $O(1)$, but which requires more computational effort on the part of the server. A third technique provides a tradeoff between the computational effort and the number of rounds. These protocols are complemented by several lower bounds and impossibility results. We prove that all of our protocols are existentially optimal in terms of the number of bits sent by the server, i.e., there are distributions for which the total number of bits exchanged has to be at least n. In addition, we show that there is no protocol that is optimal for every distribution (as opposed to just existentially optimal) in terms of bits sent by the server. We demonstrate this by proving that it is undecidable to compute (even approximately), for an arbitrary distribution D, the expected number of bits that must be exchanged by the server and client on the distribution D.

[1] This research was conducted in part while Micah Adler was at the Computer Science Department of the University of Toronto (supported by an operating grant from the Natural Sciences and Engineering Research Council of Canada, and by ITRC, an Ontario Centre of Excellence), and in part while he was at the Heinz Nixdorf Institute Graduate College, Paderborn, Germany.

[2] Bruce Maggs is supported in part by the Air Force Materiel Command (AFMC) and ARPA under Contracts F196828-93-C-0193 and F19623-96-C-0061, by ARPA Contract N00014-95-1-1246, and by an NSF National Young Investigator Award, No. CCR-94-57766, with matching funds provided by NEC Research Institute and Sun Microsystems. This research was conducted in part while he was visiting the Heinz Nixdorf Institute, with support provided by DFG-Sonderforschungsbereich 376 "Massive Parallelität: Algorithmen, Entwurfsmethoden, Anwendungen".

The views and conclusions contained here are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either express or implied, of AFMC, ARPA, CMU, or the U.S. Government.

1 Introduction

In the past two years a number of commercial networking technologies with asymmetric bandwidth capabilities have been introduced. In some cities, for example, telephone companies have started trials of asymmetric digital subscriber lines (ADSLs). In Pittsburgh, this technology provides a download speed of 1.5mbs, and an upload speed of 64kbs. In the DirecPC network provided by Hughes, a satellite beams data down to the user's home at 400kbs, and the user sends data back using an ordinary phone line (at 33.6kbs). Internet access provided through cable-television networks is also typically asymmetric. In the Boston area, for example, MediaOne is offering service with a download rate of 1.5mbs and an upload rate of 300kbs. Using ordinary telephone lines, 56k modems can download at up to 56kbs, but can upload data at a maximum rate of 33.6kbs.

Asymmetric communication scenarios also arise in situations where the bandwidth provided by the underlying communication channel is not asymmetric. For example, a mobile node connected to a base station via a wireless channel may wish to limit its transmissions in order to conserve power, while the base station may have significantly less reason to limit its power consumption. The issues and solutions in this paper apply to this and other types of asymmetry, but to keep our terminology consistent, we shall refer to the high-speed direction of sending and the low-speed direction of sending.

This paper aims to address the limitations of asymmetric network connections by examining the following question. Is it possible to use a high-speed downlink to improve the performance of a low-speed uplink? Perhaps surprisingly, in several natural situations the answer is yes. To be more precise, suppose that a client at the end of the downlink has an n-bit string x to send to a server at the end of the uplink. We show that in certain circumstances, the server can use the high-speed downlink to reduce the expected number of bits sent by the client across the low-speed uplink to significantly less than n.

1.1 Reducing the number of bits sent by the client

Classical coding techniques for reducing the expected number of bits sent across a communication channel include source codes such as Shannon codes [24] and static Huffman codes [6]. These codes can be used when x, the string to be sent, is drawn randomly from a probability distribution D that is known to both the client and the server. Using such a code, the expected number of bits that must be transmitted from the client to the server is at most $H(D)+1$, where $H(D)$ is the (binary) *entropy* of the distribution D, a quantity that varies between 0 and n, given by the equation

$$H(D) = \sum_{x \in \{0,1\}^n} D(x) \log_2 \frac{1}{D(x)},$$

where x is an n-bit string and $D(x)$ is the probability of x being drawn from the distribution D.

A drawback of source codes is that they can only make use of information about D known to both the client and the server. This drawback is particularly relevant when using source codes for asymmetric scenarios, since asymmetry can result in the server having more information about D than the client for a variety of reasons. In some circumstances the (limited bandwidth) client may lack the resources to locally store or compute the distribution D. This is an important consideration since the storage required just to represent a distribution on n-bit strings can be exponential in n. A small mobile host communicating with a base station, for example, may be more limited than the base station in terms of computational resources that can be allocated to computing or storing information about D.

It is also possible that the client is unable to determine, in an information-theoretic sense, the same information about the distribution that the server knows. This would occur, for example, when the server is collecting data items from a large number of clients, and these data items are correlated in some manner. After the server has collected a sample of the data items, it is able to approximate the distribution D of future data items to be sent, (see [7] for a good survey on techniques for this) but the clients do not gain any such information.

This paper addresses the situation where the server has information about the string being sent that is not known to the client. We study the following problem: a string x must be transmitted from the client to the server, where x is drawn from an arbitrary distribution D that is known to the server, but not the client. We assume that the only information known to the client about D is x, the current string to be sent. We shall demonstrate that in such a scenario, the presence of a high-speed downlink allows the client to transmit the string x to the server by sending an expected number of bits that is only a small constant factor more than $H(D)$. This is close to optimal, even if we compare the performance of this pessimistic scenario to the case where the client has full knowledge of the distribution D.

The protocols we introduce demonstrate that the high-speed downlink can be used to improve the performance of the low-speed uplink. Also, in all cases the memory and computational requirements of the client are minimal. Thus, our protocols can be used to reduce the computational resource requirements of the client: they enable the client to do away with locally computing and storing any information about the distribution D without a large increase in the number of bits that the client must send.

1.2 The model

We study a model based on Yao's two-party communication complexity model [27]. To enable the client to transmit its n-bit string to the server, the client and the server communicate bits to each other, as specified by some fixed protocol $\mathcal{P}$. The protocol $\mathcal{P}$ specifies at each step whether the client or the server sends the next bit, as well as the value of that bit. A bit sent by the client can only depend on the bits sent thus far by the server and the information known to the client at the start of the protocol. The analogous requirement holds for the server. Thus, the model is asynchronous: the client and server are not able to obtain information based on when the bits arrive. When the protocol terminates, the server must have enough information to determine the n-bit string x with certainty.

A *round* of the protocol is defined as a maximal sequence of consecutive bits sent by the server (without any bits sent in between by the client), followed by a maximal sequence of consecutive bits sent by the client. Minimizing the number of rounds required by a protocol is an important consideration in scenarios where there is some fixed overhead for each round, such as message latency or bits sent during handshaking protocols.

We also consider how expensive a protocol is in terms of local computation. All protocols considered in this paper are fairly minimal in the computational demands they place on the client, and thus this aspect of the protocols will not be addressed by the model. A more interesting issue is the computational demands placed on the server. We model server computation by assuming that the server has access to the distribution D via a black box. The server is allowed to query the black box for any k-bit string s, for any $0 \leq k \leq n$. The black box returns the cumulative probability of n-bit strings that start with the k-bit string s. The local computation of the server in a given protocol is defined to be the number of such black box queries performed by the protocol.

The actual computational effort required of the server in a given protocol is dependent on how the distribution D is presented to the server, and thus this black box model is not always going to give an accurate prediction of the computational effort of a protocol. However, the black box model has the advantage of simplicity. Furthermore, the accuracy of the computational effort performance measure does not affect the accuracy of any of the other performance measures. We next describe two scenarios where this black box model gives an adequate approximation to the computational requirements of obtaining information about the distribution.

In the first example, either a description of the distribution or a set of samples that defines the distribution is stored as an ordered tree. By storing cumulative information at internal nodes of the tree, black box queries can be answered by a single root to leaf traversal of the tree. In the second example, $D(x)$ is dependent only on the Hamming distance between x and some fixed string x'. This kind of distribution is useful when the server has access to a previous version of x, such as a previously transmitted video image or an outdated file. In this case, the cumulative probability of strings with a given prefix s is a function of only the length of s and how many bits in s match x'. For both examples, black box queries can be answered efficiently, and thus the black box provides an abstraction of the kind of information about D that is available to the server. Note that in the second example the server could specify any subset of the bits of x (instead of only bits in some prefix of x). All results in this paper also extend to a black box model where arbitrary subsets of the bits can be specified.

In general, we characterize a protocol in terms of four parameters, $[\sigma, \phi, \lambda, \rho]$, where σ is the expected number of bits sent by the server, ϕ is the expected number of bits sent by the client, λ is the expected number of black box queries performed by the server, and ρ is the expected number of rounds.

1.3 Our results

Shannon's theorem [24] implies a lower bound of $H(D)$ on the expected number of bits sent by the client. If the server starts by transmitting a description of the entire distribution to the client, then by using a static Huffman coding scheme [6] the string can be transferred in one round with at most $H(D) + 1$ expected bits sent by the client. However, such a scheme can be completely impractical since the number of bits the server sends to the client can be exponential in n.

We show that we can in fact do much better. We begin by describing a $[3n, 1.71H(D) + 1, 3n, 1.71H(D) + 1]$-protocol. We call this protocol **Computation-efficient** because the expected number of black-box queries performed by the server is asymptotically optimal. This protocol is useful even in scenarios where the local computation of the server is not accurately represented by the black box model: the computation required of the client is simple and efficient, our analysis provides very small constants, and most notably, the expected number of bits sent by the client is within a factor of 1.71 of the lower bound.

The drawback of the protocol **Computation-efficient** is that the expected number of rounds required is linear in the entropy of the distribution. Thus, we present a second protocol, based on a completely different technique, that uses an expected constant number of rounds. In particular, we present an $[O(n),\ O(H(D)+1), 2^n, O(1)]$-protocol, which we call **Round-efficient**. We also present a third technique, which we call **Computation-Rounds-Tradeoff**(c). This protocol achieves a tradeoff between the expected number of black box queries and the expected number of rounds required. For any positive integer c between 1 and n, **Computation-Rounds-Tradeoff**(c) provides an $[O(n), O(H(D)+1), O(\frac{n2^c}{c}), O(\min(\frac{n}{c}, H(D)+1))]$-protocol.

We also provide several interesting lower bounds. We first address the issue of how many bits must be sent by the server. We demonstrate a lower bound of n on the total number of bits that must be exchanged. This implies that in any protocol where the client sends fewer than $\frac{n}{2}$ bits, the server must send at least $\frac{n}{2}$ bits. Thus, the expected number of bits that the server sends in our protocols is within a constant factor of optimal for protocols that are efficient in terms of the expected number of bits sent by the client. We first show that for any $0 < h \leq n$, there is a class of distributions $\mathcal{D}_h$ with entropy h, such that when a distribution D_h is chosen uniformly at random from $\mathcal{D}_h$, the expected number of bits that must be exchanged is at least n. This result follows from techniques developed in [20].

There are, on the other hand, specific distributions D' where the optimum expected number of bits exchanged is $o(n)$, but for which our protocols require $\Omega(n)$ bits to be exchanged. Thus, a natural question to ask is: does there exist a protocol where the client and the server exchange close to the optimal number of bits for every distribution? We demonstrate that the answer to this question is no. This follows from a proof, using Kolmogorov complexity, that it is undecidable to compute, for an arbitrary distribution D, even what the value of the optimal number of bits is. Also, the problem remains undecidable even if only an approximate solution is required. For example, computing a value that is guaranteed to be between the optimal number of bits and Ackermann's function applied to the optimum number of bits is undecidable. This implies that although our protocols do not use the optimal number of bits for every distribution, they do provide the best possible general guarantees (up to constant factors).

We also address the issue of how many rounds of communication are required. A natural goal would be to derive a *single-round* protocol: a protocol which starts with the server sending the client some number of bits, after which the client responds with some number of bits, after which the server is guaranteed to know the string x. We demonstrate that no efficient single-round protocol can exist. Specifically, for any $0 < h \leq n$, and any single-round protocol P^S, there is a distribution D_h with $H(D_h) = h$, such that if the expected number of bits sent by the client on D_h is at most $\frac{n}{16}$, then the server must always send an exponential number of bits: $\Omega(n2^h)$.

We also show that the expected number of black-box queries performed by the server in protocol **Computation-efficient** is asymptotically optimal. In particular, we show that for any entropy h, there is a distribution D with entropy $H(D) = h$ for which the expectation of the sum of the number of bits sent by the client plus the number of black-box queries is at least n.

1.4 Previous and related work

The question of sending a string x from a client to a server, where the server has some information about x unknown to the client, has a long history in the area known as interactive communication [14, 17, 18, 19, 20, 21, 22, 25]. Here, it is typically assumed that a pair (x, y) is drawn from a joint probability distribution D_p over pairs (x, y), where D_p is known to both the client and the server in advance. The string x is given to the client, the string y is given to the server, and the task is to communicate the n-bit string x to the server. However, all of this work studies the case where the communication channel is symmetric. Thus, there is no regard to which direction bits are sent: the only objective is to minimize the total number of bits that are sent.

Furthermore, the techniques used to derive protocols in the area of interactive communication typically restrict the types of distributions that are allowed. This is not surprising given the lower bound of n on the number of bits that must be exchanged in the worst case. In the symmetric scenario, if n bits must be exchanged, the most efficient technique is to send the string x directly to the server. In the asymmetric scenario we consider, on the other hand, we have the advantage that we can send those n bits in the other (faster) direction, and thus we do not need to make any restrictions on the distribution D. As an example of the restrictions considered in the symmetric case, the protocols provided in [20] assume that the pair (x, y) is uniformly distributed over the set of pairs with nonzero probability. In our framework, this would require that in the distribution D, all inputs x that are possible occur with the same probability. Many of the other results in this area focus on the worst-case number of bits for any possible input. In this case, there is also no reason to take into account the actual probabilities of strings, further than differentiating strings that occur with probability 0 and strings that occur with probability greater than 0.

Interactive communication is part of the large body of work on two-party communication complexity. A good reference for this area is the book by Kushilevitz and Nisan [10]. Most of this work examines symmetric communication channels, and analyzes the total number of bits transmitted by the two parties, and sometimes the number of rounds. There is relatively little work on asymmetric communication complexity.

One notable exception is a body of work connecting asymmetric communication complexity to lower bounds on the time to perform operations on various data structures [15, 16]. The paper [16] is most closely related to this one. However, instead of the problem of sending information from the client to the server, they consider the problem of computing a $\{0, 1\}$-function, where a portion of the input appears at the client and a portion of the input appears at the server. They present a number of general techniques for proving tradeoffs between the number of bits sent by the server and the number of bits sent by the client, and apply these techniques to several fundamental problems, such as set membership, set disjointness, and greater than. As an example, in the set membership problem, the server holds a set S of strings, and the client holds a single string x. The value of the function is a 1 if x belongs to S and a 0 otherwise.

One way to view the issues addressed in this paper is as a combination of the work on interactive communication and the work on asymmetric communication complexity. The problem we study is similar to that studied in the interactive communication literature, and the model we use is similar to that used in the asymmetric communication complexity model.

Subsequent to the appearance of a preliminary version of this paper in [1], there has been further work on the problem considered here. Eduardo Sany Laber has shown [11] that the expected number of bits used by the client in the protocol **Computation-efficient** is even better than we show in this paper. In particular, for any distribution D, the expected number of bits used by the client is at most $1.08H(D) + 1$. Protocols that further improve on the expected number of bits sent by the client (at the cost of more bits sent by the server) appear in [26] and [23].

There have also been recent experimental studies of asymmetric bandwidth [2, 8, 9]. These studies have shown that in practice, even if the flow of data is entirely downstream, the overall rate at which data can be transferred in asymmetric networks may be limited by the upload speed. The explanation for this is that in the TCP protocol, acknowledgments must be sent upstream for all data that travels downstream, and the flow of data will stall if the acknowledgments cannot keep up.

2 Our protocols

In this section we provide three protocols. All three are within a constant factor of optimal in terms of the number of bits sent by the client, as well as the number of bits sent by the server. The first is also asymptotically optimal in terms of the number of black box queries required, the second is asymptotically optimal in terms of the expected number of rounds required, and the third allows us to achieve a tradeoff between black box queries and rounds. In many cases, the server sends the client a message of length k, where the client does not know the value of k. In scenarios where k must be explicitly specified, we use self-delimiting strings [12], which prepend the length of a message to the message. A message of length k is sent using $k + O(\log k)$ bits. The effect of these additional bits on the analysis is negligible, and is ignored.

2.1 Protocol Computation-efficient

In this protocol, the server sends the client queries consisting of candidate prefixes for the client's string, and the client responds positively or negatively to these queries. The server keeps track of the responses, and the string r held by the server in this protocol represents the prefix of x that the server has already determined. Positive responses allow the server to extend r, and negative responses allow the server to remove strings from consideration. Future queries to the client depend on the client's previous responses. In order to do this efficiently, the results of black box queries are adjusted from the *a priori* probability of a string occurring to reflect the information learned from the client thus far. Given a set of excluded strings X and p_Q, the result of a black box query Q, let P_X be the sum of the probabilities of the strings in X that are consistent with Q. We call the value $\frac{p_Q - p_X}{1 - p_X}$ the *exclusion adjusted* probability for Q. This value reflects the fact that the actual string cannot be in the set X. Note that we shall make normal black box queries, and then adjust the answers given by these black box queries to reflect the exclusion adjusted probabilities.

The protocol is defined as follows:

Let r be the empty string. Repeat the following until $r = x$:

- Conditioning on all information learned from the client thus far, the server finds a prefix of the unknown bits as follows:
  - Let s be the empty string.
  - The server repeats the following until it has a prefix s that occurs with probability between $\frac{1}{3}$ and $\frac{2}{3}$, inclusive, or that extends to the end of the string.
    - Query the black box for $rs0$.
    - If the exclusion adjusted probability of the value returned by the black box is $> \frac{2}{3}$, then a 0 is appended to the end of s.
    - If the exclusion adjusted probability of the value returned by the black box is $< \frac{1}{3}$, then a 1 is appended to s.

    * If the exclusion adjusted probability of the value returned by the black box is between $\frac{1}{3}$ and $\frac{2}{3}$, then a 0 is appended to s.

- The server sends s to the client.
- If s matches x, the client responds with a "y", after which the server sets $r = rs$.
- If s differs from x, the client responds with an "n", after which the server updates the exclusion adjusted probabilities accordingly.

Note that the prefix sent always either extends to the end of the string, or occurs with probability between $\frac{1}{3}$ and $\frac{2}{3}$, since when a prefix that occurs with probability $p > \frac{2}{3}$ is extended by one bit, the prefix with the more likely of the two settings for that bit occurs with probability at least $\frac{p}{2}$. Also note that the actions performed by both parties are deterministic. The only source of randomness is the value of x given to the client.

Theorem 1 *For any distribution D, protocol* **Computation-efficient** *is a* $[3n, 1.71H(D)+1, 3n, 1.71H(D)+1]$*-protocol.*

Proof: We first show that the expected number of bits sent by the client is $O(H(D)+1)$. For any input distribution D, we model the bits sent by the client as a tree, where each internal node of the tree corresponds to a query asked by the server, and each leaf of the tree represents a string held by the client. Each left branch of the tree represents a "y" response by the client and each right branch of the tree represents a response of "n". Upon reaching a leaf, the server and client have agreed on some string x_i, and there is exactly one leaf for every distinct string x_i. Thus, in this tree, the probability of the protocol reaching any leaf x_i is exactly $D(x_i)$.

The choice of prefix that the server sends to the client implies that at every internal node of the tree, the right branch occurs with probability $\leq \frac{2}{3}$, and the left branch either occurs with probability $\leq \frac{2}{3}$ or represents an affirmative answer to a prefix that extends to the end of the string (which is a leaf of the tree). Thus, along any path from the root to a leaf, there is at most one branch that occurs with probability $> \frac{2}{3}$. Therefore, the depth of leaf x_i is at most $1 + \log_{2/3} D(x_i)$. This implies that the expected number of bits sent by the client is at most $\sum_{x_i} D(x_i)(1 + \log_{2/3} D(x_i)) = 1 + H(D)/\log(\frac{3}{2}) \approx 1.71H(D) + 1$.

The bound on the expected number of rounds follows from the fact that the client sends one bit in each round. To see that the expected number of bits sent by the server is at most $3n$, let E_i be the *a priori* expected number of strings sent by the server to the client that include the i^{th} bit position of the string held by the client. For every prefix sent by the server, the probability of a successful match is at least $\frac{1}{3}$. Therefore, $E_i \leq 3$, and the result follows from the linearity of expectation. The bound on the number of black box queries follows from the fact that each bit sent by the server corresponds to a single black box query. ■

The next protocol uses only a constant expected number of rounds, but at the cost of a larger number of black box queries.

2.2 Protocol Round-efficient

For any distribution D, let $T(D)$ denote the strings in sorted order from most likely to occur to least likely to occur. Let $\tau(D)$ denote a partition of the strings into sets $\mathcal{X}_i$. Set $\mathcal{X}_1$ contains the first h_1 strings of $T(D)$, where h_1 is chosen so that $h_1 > 0$ and $|(\sum_{x_i \in \mathcal{X}_1} D(x_i)) - \frac{1}{2}|$ is minimized. In other words, set $\mathcal{X}_1$ contains as close to half the probability weight as possible. Set $\mathcal{X}_2$ contains the next h_2 strings, where h_2 is chosen so that $\mathcal{X}_2$ contains as close to half the remaining probability weight as possible, and similarly with the remainder of the sets in the partition. Note that the last set in the partition (denoted $\mathcal{X}_r$) contains exactly 1 string. Also note that each set $\mathcal{X}_j$ either contains only one string and contains at least $\frac{1}{3}$ of the remaining probability weight, or contains between $\frac{1}{3}$ and $\frac{2}{3}$ of the remaining probability weight. The protocol we use proceeds in phases. During phase i, we use hashing to check if the string is in the set $\mathcal{X}_i$.

The protocol uses $\mathcal{F}_n$, the family of pairwise independent hash functions where for each $F \in \mathcal{F}_n$, we have $F(x) = ax + b$, where arithmetic is with respect to the finite field $GF[2^n]$ [3]. Here, a and b are values chosen uniformly and independently at random from $GF[2^n]$, and thus the total number of bits required to describe any $F \in \mathcal{F}_n$ is $2n$. Also, note that with this construction, for any $k < n$, the first k bits of $F(x)$ also forms a pairwise independent hash function (see for example [13]).

Within our protocol, the client computes the value of $F(x)$, for some F chosen randomly by the server. Within phase i of the protocol, the client and the server check to see if $x \in \mathcal{X}_i$. This requires that the server know the first $\lceil \log h_i \rceil$ bits of $F(x)$. The variable ℓ in the protocol represents the total number of bits of $F(x)$ seen so far; if $\lceil \log h_i \rceil > \ell$, then the client must send the server more bits of $F(x)$ in order to perform phase i of the protocol. The protocol is proceeds as follows:

- The server queries the black box to find $D(x)$ for all possible strings x, and uses this information to determine the partition $\tau(D)$. To do this, the server sorts the strings based on $D(x)$.
- The server sends to the client two randomly chosen n-bit coefficients a and b for a hash function $F \in \mathcal{F}_n$.
- Let $i = 1$ and let $\ell = 0$.
- Repeat the following until x, the client's string, is known by the server.
  - The server sends to the client the binary representation of $\ell' = \lceil \log h_i \rceil$, where $h_i = |\mathcal{X}_i|$.
  - If $\ell' > \ell$, the client sends to the server bits $\ell + 1$ through ℓ' of $F(x)$. Note that this is sufficient for the server to know the first ℓ' bits of $F(x)$.
  - $\ell = \max(\ell, \ell')$.
  - The server finds all strings $x' \in \mathcal{X}_i$ such that the first ℓ bits of $F(x')$ are the same as the first ℓ bits of $F(x)$, and sends the strings to the client.
  - If the client sees its string in the list sent by the server, the client sends a "y", followed by the index of its string within the list, and the protocol terminates.
    - Otherwise, the client sends the server an "n".
  - If $i = r - 1$, then there is only one possible string remaining, and the protocol terminates.
    - Otherwise $i = i + 1$.

Theorem 2 *Protocol* **Round-efficient** *is an* $[O(n), O(H(D) + 1), 2^n, O(1)]$*-protocol.*

Proof: We first bound the expected number of bits sent by the client. We do this as follows: we introduce a code $\bar{\tau}$, called the *comparison code* for the distribution D, and show that the expected codeword length using $\bar{\tau}$ is $O(H(D)+1)$. We then show that the expected number of bits sent by the client is at most a constant factor more than the expected codeword length of $\bar{\tau}$. The use of the comparison code in this proof greatly simplifies the proof of Theorem 3.

We describe the code $\bar{\tau}$ as a tree. In this tree, every internal node represents a "y" or "n" response sent by the client in the protocol **Round-efficient** (the other bits sent by the client do not effect the code $\bar{\tau}$). Every left branch represents the transmission of a "y", every right branch represents the transmission of an "n", and every leaf represents a string. The subtree found by starting at the root, taking $0 \leq k \leq r-2$ right branches, followed by a single left branch, contains exactly the strings in $\mathcal{X}_{k+1}$. This portion of the code $\bar{\tau}$ is identical to the "y" and "n" bits sent by the client. Within each subtree, we use any code with the following property: at any internal node of the tree, either the probability of taking the left branch is between $\frac{1}{3}$ and $\frac{2}{3}$ (we call such a node a *balanced* node), or the branch with higher probability is a leaf of the tree. Examples of such codes are those defined by the bits sent by the client in protocol **Computation-efficient**, and Fano codes [4].

Let $E(\bar{\tau})$ be the expected codeword length using the code $\bar{\tau}$ on a string x_i drawn from the distribution D. Using an argument similar to the proof of Theorem 1, we show that $E(\bar{\tau}) = O(H(D)+1)$. We first point out that along any path from the root to a leaf, there can be at most one branch that occurs with probability greater than $\frac{2}{3}$. We call such a branch a *weighty* branch. Note that none of the right branches taken before entering a subtree containing the strings in a set $\mathcal{X}_i$ can be weighty. The first left branch taken (when entering the subtree) may be weighty, but only if the set $\mathcal{X}_i$ consists of a single string. In that case, the first left branch is the only weighty branch on the path to that string. If the first left branch is not weighty, then there can only be one node on the path from root to leaf that is not balanced: the last node. Again, there is at most one weighty branch on the path from root to leaf. Therefore, the depth of leaf x_i is at most $1+\log_{2/3} D(x_i)$. This implies that $E(\bar{\tau})$ is at most $\sum_{x_i} D(x_i)(1+\log_{2/3} D(x_i)) = 1 + H(D)/\log(\frac{3}{2}) \approx 1.71H(D)+1$.

We next show that $E(A)$, the expected number of bits sent by the client on the distribution D, is $O(E(\bar{\tau}))$. We first derive a lower bound for $E(\bar{\tau})$ that will be useful in proving an upper bound on $E(A)$. We assume that there is more than one string x_i such that $D(x_i) > 0$, since when this is not the case, the number of bits sent by the client can easily be seen to be $O(1)$. We derive an expression for the minimum depth of any string in $\mathcal{X}_j$ in $\bar{\tau}$. The depth of the string in $\mathcal{X}_j$ is at least 1; this suffices for the case where $h_j = 1$ (recall that $h_j = |X_j|$). When $h_j > 1$, let x_m be a minimum-depth leaf in $\mathcal{X}_j$, where in case of a tie, x_m is the leaf with lowest probability. Let r_j be the root of the subtree induced by $\mathcal{X}_j$. Since there are no leaves at a smaller depth than x_m, all the nodes on the path from r_j to x_m, with the possible exception of the last node, are balanced. Either the last node is balanced, or the branch taken from that node to reach x_m occurs with probability $> \frac{2}{3}$. Thus, every branch on the path from r_j to x_m occurs with probability $\geq \frac{1}{3}$.

Let $q_j = \sum_{x_i \in \mathcal{X}_j} D(x_i)$, the probability of reaching r_j. The length of the path from r_j to x_m is at least $\log_3 \frac{q_j}{D(x_m)}$. Let $m_j = \max_{x_i \in \mathcal{X}_j} D(x_i)$ be the maximum probability of any string that appears in $\mathcal{X}_j$. Since $m_j \geq D(x_m)$, we see that

$$E(\bar{\tau}) \geq \sum_{j=1}^{r} q_j \max(1, \log_3 \frac{q_j}{m_j}).$$

We next bound the expected number of bits sent by the client. The client sends three kinds of bits: bits that represent the image of a hash function, bits that represent a "y" or "n" answer to a list of strings sent by the server, and, after a "y" answer, bits that represent the index of the correct string within that list. The index is only sent once. Since we have a pairwise independent hash function, the probability that any given string hashes to the same value as x is $\frac{1}{h_i}$ when x is in the set $\mathcal{X}_j$. By linearity of expectation, the expected number of collisions with the client's string is $\frac{h_i-1}{h_i}$, and so the expected number of strings in the list is less than 2. Thus, the total expected number of index bits is 1. When the client finds out that the string it holds is not in any of the first $j-1$ sets $\mathcal{X}_1 \ldots \mathcal{X}_{j-1}$, it may be required to transmit some additional bits of the hash function image, but never more than $\lceil \log h_j \rceil$ additional bits. This occurs with probability $s_i = 1 - \sum_{j=1}^{i-1} q_j$, where we define $s_1 = 1$. Thus, the expected number of bits sent by the client is at most

$$E(A) = \sum_{i=1}^{r-1} s_i(\log h_i + O(1)).$$

Here, the $O(1)$ term accounts for the index bits, the "y" or "n" bits, as well as the rounding of $\log h_i$. In order to compare this expression with that derived for $E(\bar{\tau})$, we use the following facts:

1. $q_i \geq \frac{s_i}{3}$
2. When $h_i > 1$, $s_i \leq 9q_{i+1}$.
3. When $h_i > 1$, $h_i \leq \frac{6q_{i+1}}{m_{i+1}}$.

Fact 1 follows directly from the fact that in constructing the set $\mathcal{X}_j$, we used as close to half the remaining probability weight as possible. When $h_i > 1$, we see that since the strings are partitioned in order from most weight to least weight, $q_i \leq \frac{2}{3}s_i$. Thus, since $s_{i+1} = s_i - q_i$, $s_{i+1} \geq \frac{s_i}{3}$. By Fact 1, $q_{i+1} \geq \frac{s_{i+1}}{3}$, which gives us Fact 2. To prove Fact 3, note that since every string in the set $\mathcal{X}_i$ occurs with greater probability than any string in the set $\mathcal{X}_{i+1}$, we have $m_{i+1} \leq \frac{q_i}{h_i}$. Since $q_i \leq \frac{2}{3}s_i$ combined with Fact 2 implies that $q_i \leq 6q_{i+1}$, Fact 3 follows.

To apply these Facts to $E(A)$, there are two cases for each term of the summation. When $h_i = 1$, then Fact 1 implies that $s_i(\log h_i + O(1)) = O(q_i)$. In the case that $h_i > 1$, Facts 2 and 3 give us that $s_i(\log h_i + O(1)) \leq 9q_{i+1}(\log \frac{6q_{i+1}}{m_{i+1}} + O(1))$. Combining both cases, we see that

$$s_i(\log h_i + O(1)) \leq 9q_{i+1}(\log \frac{6q_{i+1}}{m_{i+1}} + O(1)) + O(q_i).$$

This implies that

$$E(A) = \sum_{i=1}^{r} O(q_i(\log \frac{q_i}{m_i} + 1)).$$

This implies that $E(A) = O(E(\bar{\tau}))$, which in turn implies that $E(A) = O(H(D) + 1)$.

The expected number of rounds required by this protocol is 6, which follows from the fact that to process each set $\mathcal{X}_i$, only 2 rounds are required. Conditioned on the fact that no previous set has contained the string held by the client, each set contains this string with probability at least $\frac{1}{3}$, and thus the expected number of sets $\mathcal{X}_i$ that must be processed is 3.

The server sends three kinds of bits to the client: bits that represent the number $\lceil \log h_i \rceil$, bits that describe the hash function to be used, and bits that represent strings that map to the same image of the hash function as x. For any set $\mathcal{H}_j$, the number of bits required to represent $\lceil \log h_j \rceil$ is $\log \log h_j + o(\log \log h_j) < \log n + o(\log n)$. The number of bits required to describe the hash function is $2n$. Since we have a pairwise independent hash function, for each examined set $\mathcal{X}_i$, the expected number of strings that map to the same image as x_i, not counting x_i itself, is at most one. The expected number of sets $\mathcal{X}_i$ examined is 3, and thus the expected total number of bits representing strings other than the string x is $3n$. In addition, the string x is sent when processing the last set. Thus, the total expected number of bits sent by the server is $6n + o(n)$. ∎

Improvements. We also point out that although the constants provided by this proof are larger than the constants we provide for protocol **Computation-efficient**, in the case that for all x_i, $D(x_i)$ is an inverse power of 2, protocol **Round-efficient** can be made into a $[(3+\epsilon)n + o(n), 4H(D) + O(1+\log\frac{1}{\epsilon}), 2^n, 3]$-protocol, for any $0 < \epsilon \leq 1$. Furthermore, if a shared source of randomness is allowed (i.e., if the hash function is chosen beforehand), then this can be further improved to a $[(1+\epsilon)n + o(n), 4H(D) + O(1+\log\frac{1}{\epsilon}), 2^n, 3]$-protocol, for any $0 < \epsilon \leq 1$.

We first describe how to improve the expected number of server bits to these values. Without a shared source of randomness, the server still sends the two randomly chosen n-bit coefficients a and b, i.e., $2n$ bits. In the last round, the server sends the n-bit string for a total of $3n$ bits. The ϵn bits are used to send strings that collide with the client's string (i.e., have the same hash value.) In the original protocol, we assumed that there would be one collision per round, i.e., n bits per round. But by using a slightly larger hash table, we can reduce the expected number of collisions. In particular, for every bit that we add to the size of the hash value, the size of the hash table doubles, and the chance of a collision falls in half. Thus, by adding $\log\frac{1}{\epsilon} + 2$ bits to the size of the hash value, we can get the expected number of collisions down to ϵ over the entire protocol.

The bound on the expected number of bits sent by the client relies on the fact that probabilities are inverse powers of two. In this case, each set $\mathcal{X}_i$ contains exactly $\frac{1}{2}$ of the remaining probability weight. This changes the bounds in Facts 1, 2, and 3 as follows:

1. $q_i \geq \frac{s_i}{2}$
2. When $h_i > 1$, $s_i \leq 4q_{i+1}$.
3. When $h_i > 1$, $h_i \leq \frac{2q_{i+1}}{m_{i+1}}$.

Also, there must be a comparison code in which all nodes are perfect $\frac{1}{2}$ - $\frac{1}{2}$ splits, so that our lower bound on $E(\bar{\tau})$ becomes

$$E(\bar{\tau}) \geq \sum_{j=1}^{r} q_j \max(1, \log_2 \frac{q_j}{m_j}).$$

Otherwise, the analysis of the expected number of bits sent by the client follows along the lines of the proof of Theorem 2.

For the bound of 3 on the number of rounds, note first that we can guarantee 4 expected rounds easily, since each set $\mathcal{X}_i$ requires 2 rounds to process, and when each set $\mathcal{X}_i$ contains exactly $\frac{1}{2}$ of the remaining probability weight, the expected number of sets examined is 2. This can be improved to 3 by combining the second round of the process for examining the set $\mathcal{X}_i$ with the first round for examining the set $\mathcal{X}_{i+1}$.

Furthermore, we can guarantee that the server never sends more than $O(n)$ bits (instead of just $O(n)$ bits in expectation) without changing the other performance measures by more than a constant factor. We process the set $\mathcal{X}_i$ as follows: if $h_i = |\mathcal{X}_i| \geq n$, then we increase the number of hash bits used from $\lceil \log h_i \rceil$ to $\lceil 3 \log h_i \rceil$. This increases the total number of client bits by no more than a factor of 3, and ensures that the probability that any string in $\mathcal{X}_i - x$ hashes to the same value as x is at most $\frac{1}{n^2}$. After $O(\log n)$ sets $\mathcal{X}_i$ have been processed, the total probability of the remaining possible strings is less than $\frac{1}{n}$, and so in this case the client can send x in its entirety. Thus, the probability that there is any set $\mathcal{X}_i$ considered in the protocol with a string in $\mathcal{X}_i - x$ that hashes to the same value as x is at most $\frac{1}{n}$. This means that if the candidate string sent from the server to the client does not match x, the total contribution, to the expected number of bits sent by the client, of sending the entire string x is $O(1)$.

For this technique to ensure that the probability of a string in $\mathcal{X}_i - x$ hashing to the same value as x is no larger than $1/n$, the client must send $\Omega(\log n)$ bits to the server. However, when $h_i \ll n$, this may require the client to send too many bits. Thus, in the case where $h_i < n$, we send the additional bits in the other direction. In particular, the server, prior to sending a candidate string to the client, hashes it to a $\lceil 3 \log n \rceil$-bit value, and sends the hashed value and the hash function to the client. The client informs the server if x hashes to the same value, and the server only then sends the actual candidate string. The probability that x hashes to the same value as any string in $\mathcal{X}_i$ is less than $\frac{1}{n^2}$, and so if x does not agree with a candidate string that has been sent, the client can again send x in its entirety. Since we only need to process $O(\log n)$ sets $\mathcal{X}_i$, the total bits sent by the server for this additional hash function is $o(n)$. Also, the total number of additional bits sent by the client is one per round.

Neither of the previous protocols are optimal in terms of both the number of black box queries required and the number of rounds required. We next show that we can smoothly trade off between the number of black box queries and the number of rounds.

2.3 Protocol Computation-Rounds-Tradeoff(c)

For c a positive integer between 1 and n, if $H(D) < \frac{n}{c}$, use protocol **Computation-efficient**. Otherwise repeat the following until the server knows the entire string:

- Conditioning on all information known thus far, the server finds a prefix of the unknown bits that either occurs with probability between $\frac{1}{3}$ and $\frac{2}{3}$, or, if that is not possible, extends to the end of the string.
- If the length of this prefix is $\leq c$ and the prefix does not extend to the end of the string, then protocol **Round-efficient** is used to determine the next c unknown bits, where probabilities are conditioned on all information transmitted by the client so far.
- Otherwise, the server sends the prefix to the client.
  - If the prefix matches the client's string exactly, the client responds with a "y"; otherwise the client responds with an "n".

Theorem 3 *Protocol* **Computation-Rounds-Tradeoff(c)** *is an* $[O(n), O((H(D)), O(\frac{n2^c}{c}), O(\min(H(D)+1, \frac{n}{c}))]$*-protocol.*

Proof: We first show that the expected number of bits sent by the client is $O(H(D)+1)$. The bits sent by the client define a tree ν. We compare the expected codeword length of ν to the expected codeword length of a related code $\bar{\nu}$ for the distribution D. In order to define $\bar{\nu}$, we first need to define some notation. For a given distribution D, let $k_1, \ldots, k_z$ be some canonical ordering of all possible calls to **Round-efficient** over all possible executions of the protocol when the server has distribution D. In k_i, there is some distribution D_i on the c bits to be determined, where D_i depends on D, and on what information about the string held by the client has been determined by the server prior to the call k_i. Let τ_i be the subset of the nodes of ν that can be reached during call k_i on some string held by the client. Note that the τ_i's are disjoint.

Let $\bar{\tau}_i$ be the comparison code (as defined in the proof of Theorem 2) for the distribution D_i. The code $\bar{\nu}$ is produced by starting with the code ν, and replacing each set of nodes τ_i with the comparison code $\bar{\tau}_i$. The nodes of ν that are descendents of the leaf of τ_i representing the c-bit string x_j become descendents of the leaf in $\bar{\tau}_i$ that also represents x_j. We saw in the proof of Theorem 2 that the expected height of any tree τ_i is at most a constant factor larger than that of the corresponding tree $\bar{\tau}_i$, and thus the expected codeword length of the code ν is at most a constant factor larger than the expected codeword length of $\bar{\nu}$.

We next show that the expected number of bits used in the code $\bar{\nu}$ is $O(H(D)+1)$. In the proof of Theorem 1, we saw that if there was at most one unbalanced node on the path from the root to any leaf in the tree representing a code, then the expected codeword length of that code is $O(H(D)+1)$. The proof here is complicated by the fact that a path may pass through one unbalanced node for each set of nodes $\bar{\tau}_i$ that it passes through.

However, we only make a call to **Round-efficient** if we have found a prefix of the c bits in question that occurs with probability at most $\frac{2}{3}$. This implies that given that we enter $\bar{\tau}_i$, the maximum likelihood leaf of $\bar{\tau}_i$ occurs with probability $\leq \frac{2}{3}$. This means that for all i, the root node of $\bar{\tau}_i$ is balanced. This in turn implies that on any path of length l from the root of $\bar{\nu}$ to a leaf of $\bar{\nu}$, the only way to have two consecutive unbalanced nodes is if the second node corresponds to a prefix of the unknown bits that occurs with probability greater than $\frac{2}{3}$ and extends to the end of the string. However, the higher probability branch of such a node must be a leaf. Thus, for a path of length l, the number of unbalanced nodes is at most $\lceil \frac{l}{2} \rceil + 1$. The number of balanced nodes on any path from the root to a leaf x_i is at most $\log_{2/3} D(x_i)$, and thus the length of the path to x_i is $O(\log \frac{1}{D(x_i)} + 1)$. It follows that the expected number of bits used in the code $\bar{\nu}$ is $O(H(D)+1)$.

To see that the expected number of bits sent by the server is $O(n)$, it is easy to bound the expected number of bits sent by calls to **Round-efficient**, and by the remainder of the protocol separately, using the techniques developed in the proofs of Theorems 2 and 1 respectively. Specifically, the expected number of bits transmitted by the server when using **Computation-efficient** is $O(1)$ for each bit of the string held by the client, for a total of $O(n)$. Also, for each use of **Round-efficient**, the expected number of bits sent by the server is O(c), and there can be at most $\frac{n}{c}$ uses of this protocol.

The bound on the number of black box queries follows from the fact that the expected number of black box queries used to determine prefixes of the string is at most $O(n)$, and the expected number of black box queries used for each of at most $\frac{n}{c}$ calls to **Round-efficient** is at most 2^c. Since $\frac{n2^c}{c} \geq n$, the number of black box queries is $O(\frac{n2^c}{c})$.

To bound the number of rounds required, note that this number can never be larger than the number of bits sent by the client, and hence the $O(H(D))$ term. The total expected number of rounds required for all calls to **Round-efficient** is at most $O(n/c)$. Since each of the prefixes either has length at least c or extends to the end of the string, and each one is a success with probability at least $\frac{1}{3}$, the expected number of prefixes sent is also at most $O(n/c)$. Note that the expected number of prefixes extending to the end of the string is $O(1)$, since each one is a success with probability $\geq \frac{1}{3}$. ■

3 Lower bounds on the number of bits sent

Shannon's Theorem directly gives us a lower bound on the expected number of bits sent by the client. Note that Shannon's Theorem typically assumes that (a) both the client and the server know D, and (b) the communication is only one-way (from the client to the server). However, if the client knows D in advance, then the server has no information that is not known to the client, and so the channel from the server to the client never needs to be used. Thus, Shannon's Theorem still applies if the client knows D, but communication is two-way. If the client does not know D at the start of the protocol, then the expected number bits sent by the client cannot be less than the expected number of bits sent when the client does know D, and thus Shannon's Theorem still holds in our scenario. This gives us the following:

Theorem 4 (Shannon [24]) *For any distribution D, the expected number of bits sent by the client is at least $H(D)$.*

For all our protocols the expected number of bits sent by the client is $O(H(D))$, and thus Theorem 4 implies that our protocols are optimal in terms of this measure. Shannon's lower bound holds even if both the client and the server know the distribution. In our scenario, only the server knows the distribution, and this can only increase the number of bits required. In the lower bounds proved from this point forward, it will be crucial that the client does not know the distribution.

We next prove a lower bound on the number of bits that must be sent by the server. To do this, we show that when the distribution is chosen from a broad class of distributions, the expected total number of bits that must be sent is at least n. This demonstrates that all of our algorithms are existentially optimal, in terms of the number of bits sent by the server, for any protocol where the client sends $\leq \frac{n}{2}$ bits.

Definition 1 *A distribution D over strings $\{0,1\}^n$ is* onto*, if for any string $x_i \in \{0,1\}^n$, $D(x_i) > 0$. A set of distributions $\mathcal{D}$ is onto if every distribution $D \in \mathcal{D}$ is onto.*

Definition 2 *A set of distributions $\mathcal{D}$ is* balanced *if, when D is chosen uniformly from $\mathcal{D}$, and then x is chosen using the distribution D, the* a priori *distribution on x is uniform.*

Theorem 5 *For any protocol P, if a distribution D is chosen uniformly at random from any onto and balanced set of distributions, the expected total number of bits exchanged by the client and the server using P is at least n.*

Proof: The proof follows from the following Theorem:

Theorem 6 (Orlitsky [20]) *For any protocol P, if the distribution D on the string x is chosen randomly from any set of distributions $\mathcal{D}$, where for every $D \in \mathcal{D}$, the set $\{x \text{ s.t. } D(x) > 0\}$ is the same, then the total number of bits that must be exchanged for the client to communicate x to the server using P is the entropy of the distribution on x prior to the choice of the distribution D.* ■

Note that for any h, we can construct a balanced and onto set of distributions $\mathcal{D}_h$ such that every $D \in \mathcal{D}_h$ has entropy exactly h. An example of such a set contains 2^n distributions on n-bit strings, where in the i^{th} distribution, the string with the n-bit binary representation of i has probability p occurring, and the remainder of the strings are distributed uniformly. By setting p appropriately, such a distribution has entropy h, for any h, $0 < h \leq n$. This gives us the following.

Corollary 1 *For any protocol P and entropy h, $0 < h \leq n$, there is a distribution with entropy h such that the expected number of bits that must be exchanged between the client and the server is at least n.*

3.1 Minimizing the Number of Bits Sent

Theorem 5 shows that the protocols we have presented are existentially optimal. That is, for many natural sets of distributions given to the server, the protocols are in fact optimal. The whole picture, on the other hand, is more involved. Distributions do exist where the optimal number of bits exchanged between the client and the server is actually much smaller than n. For example, in any distribution where only two strings are possible, it is sufficient for the server to send a $\log n$-bit description of a bit position where the strings differ, and the client to send the value of that bit.[1] Thus, none of the protocols we have presented are guaranteed to use the optimal total number of bits, or even within a constant factor of the optimal number of bits, on these distributions.

A natural question to ask is: does there exist a protocol that uses the optimal number of bits on every distribution? We next show that this is not possible. In fact, we show that any function that provides even a non-trivial approximation to the optimal number of bits for every distribution D is not even recursive! Thus, although our protocols are guaranteed to be optimal only for broad classes of distributions and not for all distributions, our protocols provide the best type of general guarantee possible.

As a means of comparison to the optimal protocol, we use the following type of protocol: based on the distribution D, the server sends to the client a description of a Turing machine $M(D)$. The client simulates running this Turing machine with the input x (the client's string), and when the Turing machine halts, the client responds to the server with the string of bits remaining on the tape. We require that $M(D)$ always halts and that at the end of the protocol the server always knows the string x. We define $OPT_{desc}(D)$ as the minimum, over all possible Turing machines for $M(D)$, of the total expected number of bits exchanged between the client and the server by such a protocol. We use the same fixed format for describing $M(D)$ for all distributions.

Note that for every distribution D, $OPT_{desc}(D) \leq n + O(1)$, since the server can always send the client a Turing machine that halts immediately, causing the client to send its string without any encoding. It is possible that there are distributions where using multiple rounds or more complicated computation at the client can lead to a protocol that exchanges less than $OPT_{desc}(D)$ bits. Nevertheless, in order for any fixed protocol to use close to the optimal number of bits for every distribution, the protocol cannot use much more than $OPT_{desc}(D)$ on any distribution. We shall demonstrate that this is not possible, even if we define "much more than" quite loosely.

[1]Note that this example motivates the onto requirement in Theorem 5. The set of distributions must also be balanced, since otherwise the client and server could agree on a static Huffman encoding based on the *a priori* distribution.

We show that $OPT_{desc}(D)$, or even any approximation to $OPT_{desc}(D)$, is not a recursive function. For such a proof, we use Kolmogorov complexity: the following proof uses the technique developed by Kolmogorov (see for example [12]) to show that the Kolmogorov complexity of a string is not a recursive function. Recall that the Kolmogorov complexity of a string x, which we denote $\mathcal{K}(x)$, is the minimum description length of the string x. We define $\mathcal{K}(x)$ to be the size of the smallest description of a Turing machine with a work tape but no input tape that can produce the string x on an output tape. We also define $\mathcal{K}(i)$, for an integer i, to be $\mathcal{K}(x_i)$, where x_i is the string representing i in binary.

Our proof also uses the set of distributions $\mathcal{D}$, which contains a distribution D_i for each integer $i \geq 2$. D_i is the distribution over $n = \lfloor \log i \rfloor$-bit strings where the string corresponding to the binary representation of $j = i - 2^{\lfloor \log i \rfloor}$ occurs with probability $1 - 2^{-2\lfloor \log i \rfloor}$. All other strings occur with equal probability. For any $i \geq 2$, $H(D_i) < 1$. Also, note that there cannot be two distributions, D_i and D_j on strings of length n, such that the transcripts of bits exchanged by the client and the server are identical when the client has the most likely string in each of the distributions. Thus, a simple counting argument gives us that for any integer m, there is always some i such that $OPT_{desc}(D_i) \geq m$.

Theorem 7 *Let $f(D)$ be any function from distributions D to $\Re$ such that $f(D) \leq OPT_{desc}(D)$, and for any integer m, there is some integer i such that $f(D_i) \geq m$. No such function $f(D)$ is recursive.*

Proof: Let $f(D)$ be any such function. We assume that $f(D)$ is recursive and reach a contradiction. Using the distributions D_i, and the function f, we define a new function, $F(m)$, defined for any natural number m. $F(m)$ is the smallest i such that $f(D_i) \geq m$. $F(m)$ is well defined by the assumption that there is always some integer i such that $f(D_i) \geq m$.

Claim 1 *$\mathcal{K}(F(m)) \geq m - c$, for some constant c.*

Proof (of Claim): By our construction, $OPT_{desc}(D_{F(m)}) \geq m$, so it suffices to show that for all i, $\mathcal{K}(i) \geq OPT_{desc}(D_i) - c$ for some constant c. This follows from the fact that $OPT_{desc}(D_i)$ is at most the expected number of bits used in the following protocol: if the distribution received by the server is one of the D_i, the server sends the client a Turing machine M that first produces the string i on a work tape. M then converts i to the string $j = i - 2^{\lfloor \log i \rfloor}$ and this is then compared to the input string x. If the two strings are equal, then the input tape is erased and replaced with a 1, after which M halts. Otherwise, the entire contents of the input tape is shifted one position right, and a 0 is added to the beginning of the tape, after which M halts. This is sufficient to inform the server of the client's string, and the expected number of client bits used is less than two. When the distribution received by the server is not one of the D_i, the server sends the client any other Turing machine that leads to a valid protocol. In such a protocol, when the server has a distribution D_i, we can minimize the expected total number of bits exchanged by defining M so that the portion of M used to produce the string i has description size $\mathcal{K}(i)$, and the description of the remainder of M has constant size. Thus, the total expected bits exchanged is at most $\mathcal{K}(i) + c$, for a constant c. ■

However, by the assumption that $f(D_i)$ is recursive, we can describe the string $F(m)$ simply by the value m. This is sufficient to determine $F(m)$, since we can compute for each i, in increasing order, $f(D_i)$ until we find the first i such that $f(D_i) \geq m$. Thus, $\mathcal{K}(F(m)) \leq \log m + c'$, for some constant c'. Since $F(m)$ is defined for all natural numbers m, we have reached a contradiction. ■

For any protocol P, let $P(D)$ be the expected number of bits used by P on the distribution D. For nonnegative integers i, let $\alpha(i)$ be $A(i,i)$, where A is Ackermann's function (see [5], p. 175). Let α^{-1} be the inverse of α, defined here as $\alpha^{-1}(j)$ is the minimum nonnegative integer i such that $\alpha(i) \geq j$.

Corollary 2 *There does not exist any fixed protocol P, such that for all D, $P(D) \leq \alpha(OPT_{desc}(D))$.*

Proof: If any such protocol P exists, then we can use the protocol to compute the function $P(D)$. If we define the function $f(D) = \alpha^{-1}(P(D))$, then we can also use the protocol P to compute $f(D)$. However, $f(D) \leq \alpha^{-1}(\alpha(OPT_{desc}(D))) \leq OPT_{desc}(D)$. Let B_n be the set of all 2^n distributions D_i over n-bit strings. The set B_n is both balanced and onto, and thus, by Theorem 5, for the protocol P, there is some i such that $P(D_i) \geq n$. Thus, for any m, there is some i such that $f(D_i) \geq m$. This implies, by Theorem 7, that the function $f(D_i)$ is not recursive, which in turn implies that there can be no such protocol P. ■

Corollary 3 *There does not exist any fixed protocol P', such that P' uses $n-1$ bits for distributions D on n bit strings where $OPT_{desc}(D) \leq \alpha^{-1}(n)$.*

In other words, we cannot guarantee to use only 1 bit less than the existential lower bound given in Theorem 5, even if we only guarantee that we do so for those distribution where the best possible protocol uses much less than n bits.

Proof: If any such protocol P' exists, then we can use P' to define (and compute) the function $f'(D)$, where if P' uses $\leq n-1$ expected bits on the distribution D over n-bit strings, then $f'(D) = 0$, and if P' uses $> n-1$ expected bits on a distribution D over n-bit strings, then $f'(D) = \alpha^{-1}(n)$. It must be the case that $f'(D) \leq OPT_{desc}(D)$. Furthermore, by Theorem 5, there is some distribution D_i in the set of distributions B_n (as defined in the proof of the previous corollary), such that $P'(D_i) \geq n$. This implies that for any integer m, there is some i such that $f(D_i) \geq m$. This implies that the function $f'(D)$ is not recursive, and thus no such protocol P' can exist. ■

4 Lower bounds on computation and rounds

We show that any protocol that requires the client to send less than $\frac{n}{2}$ bits, the server must make at least $\frac{n}{2}$ black box queries. Thus, protocol **Computation-efficient** is existentially within a constant factor of the best possible in terms of the number of black box queries required, for any protocol that does not require a large number of bits to be sent by client.

Theorem 8 *For any entropy h, there is a set of distributions $\mathcal{B}_h$, all with entropy h, such that for any protocol P, when D is chosen uniformly at random from $\mathcal{B}_h$, the number of bits sent by the client plus the number of black box queries performed by the server is at least n.*

Proof: The set $\mathcal{B}_h$ consists of 2^n distributions: one for each n bit string x_i. In the i^{th} distribution D_i, string x_i occurs with probability p, and all remaining strings occur with probability $\frac{1-p}{2^n-1}$, where p is chosen so that the resulting entropy of D_i is exactly h. For this set of distributions, the response to any black box query is always one of two results, both of which are known by the server *a priori*, provided that the server knows the set of distributions being used. Specifically, if the query specifies any k bit prefix (leaving $n-k$ bits unspecified), then the two possible answers are $2^{n-k} \cdot \frac{1-p}{2^n-1}$ and $(2^{n-k}-1) \cdot \frac{1-p}{2^n-1} + p$. The first occurs in the case where the single likely string does not match the query, and the second in the case where it does.

The actions of the server can be viewed as a binary decision tree, where each node of the tree represents either a bit received from the client or a black box query, and each leaf represents an output produced by the server. The client is equally likely to hold each of the 2^n possible strings, and each string must result in the server reaching a different leaf in the decision tree. Thus, the expected height of the leaf reached must be at least n. ■

We next turn to the question of how many rounds are required for a protocol to be efficient in terms of both client bits and server bits. The expected number of rounds required by Protocol **Round-efficient** is trivially within a constant factor of optimal. Is it possible to do better than **Round-efficient**? The best that we could hope for is a *single-round protocol*: a protocol where the server sends some number of bits to the client, and the client responds with some number of bits back to the server, at which time the server knows the string held by the client. Next, we demonstrate that any single round protocol will require either the expected number of bits sent by the client to be much larger than the minimum, or the expected number of bits sent by the server to be much larger than the minimum.

Theorem 9 *Let h be any entropy and let P be any single-round protocol where the expected number of bits sent by the client is at most $c \cdot h$, where $c \cdot h \leq \frac{n}{20}$. There is a set of distributions $\mathcal{B}'_h$, all with entropy h, such that when D is chosen uniformly at random from $\mathcal{B}'_h$, the expected number of bits sent by the server using P is at least $\frac{9}{40}n2^h$.*

Proof: By Theorem 4, we can assume that $c \geq 1$. We show that the Theorem is true for the following set of distributions $\mathcal{B}'_h$. There are $\binom{2^n}{2^k}$ distributions, $k = \lfloor h-1 \rfloor$, one for each subset of 2^k of the 2^n strings. Call such a subset a *likely subset*. For each distribution, the chosen string is one of the strings in the likely subset with probability p, and one of the other strings with the remaining probability. The strings in the likely subset each occur with equal probability, as do the strings not in the likely subset. The value p is chosen so that the entropy of each distribution is exactly h: $p \approx 1 - \frac{1}{n}$.

We show that any single-round protocol P for this set of distributions where the server sends a small number of bits would imply a protocol for the following problem that violates an easy lower bound for that problem.

Definition 3 *The* subset identification *problem: the server has an M-bit binary string I, containing exactly m 1's, where M and m are known in advance by both the server and the client. The server is allowed to send bits to the client, but no bits pass in the other direction. The task is to inform the client of the string I.*

Note that since there are $\binom{M}{m}$ possible inputs to the problem, the average number of bits sent by the server to the client, over all possible inputs, must be at least $\log \binom{M}{m}$.

Lemma 1 *Any single-round protocol A for the set of distributions $\mathcal{B}'_h$, where the expected number of bits the server sends to the client is at most b, and the expected number of bits the client sends to the server is at most $c \cdot h \leq \frac{n}{20}$, implies a protocol B for the subset identification problem with $M = 2^n$ and $m = 2^k$, where the expected number of bits the server sends to the client is at most $b + \frac{1}{2}n2^k$.*

Using the facts that $\binom{M}{m} \geq \left(\frac{M}{m}\right)^m$, and that $k \leq \frac{n}{20}$ we see that when $M = 2^n$ and $m = 2^k$, $\log \binom{M}{m} \geq \frac{19}{20}n2^k$. This gives us $b \geq \frac{9}{20}n2^k$. Since $k = h - 1$, this gives us $b \geq \frac{9}{40}n2^h$. Thus, this lemma directly implies the Theorem.

Proof (of Lemma): The protocol B proceeds on any given input as follows. The server examines the $M = 2^n$-bit input string I and determines the $m = 2^k$ bits that are set to 1. The server then sends the bits to the client that would be sent to the client during protocol A with the distribution that has the likely subset containing the strings corresponding to the location of the 1's in I.

The client and the server then separately determine the same 2^n-bit string I', that is an approximation to the string I. I' is determined as follows: for each of the 2^n n-bit strings x_i, if in protocol A, the client responds with at most $5ch$ bits, then the ith bit in I' is set to a 1, and otherwise it is set to a 0. The server then sends the client enough information to correct I' to I, which can be done efficiently because of the following:

Claim 2 *The number of 1's in I' is at most 2^{5ch}. Furthermore, the number of bits that are 1's in I that are not 1's in I' is at most $\frac{2^k}{4}$.*

Proof (of Claim): Since the server always knows what string the client has at the end of protocol A, and it is possible for the client to have every string, the client must send a different set of bits on each string. Thus, the number of strings where the client sends at most $5ch$ bits is at most 2^{5ch}. Since the expected number of bits that the client sends is at most ch, the expected number of bits sent by the client, given that the string is a likely string, is at most $\frac{1}{p}ch$. For $n > 5$, this is less than $\frac{5}{4}ch$. Thus, by Markov's inequality, the fraction of likely strings where the client sends more than $5ch$ bits is $< \frac{1}{4}$. ■

It is sufficient for the server to send the client the location of the 1's in I that are 0's in I', and the location of the 1's in I within the 1's in I'. The former requires at most $\frac{2^k}{4} \log 2^n = \frac{1}{4}n2^k$ bits, and the latter requires at most $2^k \log 2^{5ch} \leq \frac{1}{4}n2^k$ bits. This is at most $\frac{1}{2}n2^k$ bits. ■ ■

References

[1] M. Adler and B. Maggs. Protocols for asymmetric communication channels. In 39^{th} *IEEE Symposium on Foundations of Computer Science*, pages 522–533, 1998.

[2] H. Balakrishnan, V. N. Padmanabhan, and R. H. Katz. The effects of asymmetry on TCP performance. In *Proceedings of the 3rd ACM/IEEE International Conference on Mobile Computing and Networking*, September 1997.

[3] J.L. Carter and M.N. Wegman. Universal classes of hash functions. *Journal of Computer and System Sciences*, 18:143–154, 1979.

[4] R.M. Fano. *Transmission of Information.* MIT Press, Cambridge, Mass., 1961.

[5] J. Hopcroft and J. Ullman. *Introduction to Automata Theory, Languages, and Computation.* Addison-Wesley, Reading, Massachusetts, 1979.

[6] D.A. Huffman. A method for the construction of minimum redundancy codes. In *Proc. IRE*, volume 40:10, pages 1099–1101, 1952.

[7] A.J. Izenman. Recent developments in nonparametric density estimation. *Journal of the American Statistical Association*, 86(413):205–224, 1991.

[8] L. Kalampoukas, A. Varma, and K. K. Ramakrishnan. Perfomance of two-way TCP traffic over asymmetric access links. In *Proc. of Interop'97 Engineers' Conference*, 1997.

[9] L. Kalampoukas, A. Varma, and K. K. Ramakrishnan. Improving TCP throughput over two-way asymmetric links: analysis and solutions. In *Proc. of Sigmetrics*, 1998.

[10] E. Kushilevitz and N. Nisan. *Communication Complexity.* Cambridge University Press, 1997.

[11] E. S. Laber. Personal communication, `http://www.cos.ufrj.br/~ laber/`. 2000.

[12] M. Li and P.M.B. Vitanyi. Kolmogorov complexity and its applications. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science, Volume A.* Elsevier Science Publishers B.V., Amsterdam, The Netherlands, 1990.

[13] M. Luby and A. Wigderson. Pairwise independence and derandomization. Technical Report ICSI TR-95-035, Internation Computer Science Institute, Berkeley, CA, 1995.

[14] A. Orlitsky M. Naor and P. Shor. Three results on interactive communication. *IEEE Trans. on Information Theory*, 39(5):1608–1615, 1993.

[15] P. B. Miltersen. Lower bounds for union-split-find related problems on random access machnies. In 26^{th} *ACM Symposium on Theory of Computing*, pages 625–634, 1994.

[16] P. B. Miltersen, N. Nisan, S. Safra, and A. Wigderson. On data structures and asymmetric communication complexity. In 27^{th} *ACM Symposium on Theory of Computing*, pages 103–111, 1995.

[17] A. Orlitsky. Worst-case interactive communication I: two messages are almost optimal. *IEEE Trans. on Information Theory*, 36(5):1111–1126, 1990.

[18] A. Orlitsky. Interactive communication: balanced distributions, correlated files, and average-case complexity. In *Proc. 32nd IEEE Symposium on Foundations of Computer Science*, pages 228–238, 1991.

[19] A. Orlitsky. Worst-case interactive communication II: two messages are not optimal. *IEEE Trans. on Information Theory*, 37(4):995–1005, 1991.

[20] A. Orlitsky. Average-case interactive communication. *IEEE Trans. on Information Theory*, 38(4):1534–1547, 1992.

[21] A. Orlitsky. Interactive communication of balanced distributions and of correlated files. *SIAM J. of Dis. Math*, 6(4):548–564, 1993.

[22] A. Orlitsky and A. El Gamal. Interactive data compression. In 25^{th} *IEEE Symposium on Foundations of Computer Science*, pages 100–108, 1984.

[23] A. Saberi S. Ghazizadeh, M. Ghodsi. *Protocols for Asymmetric Communication Channels: Reaching the Lower Bounds.* Manuscript, Sharif University of Technology, 2000.

[24] C.E. Shannon. A mathematical theory of communication. *Bell System Technical Journal*, 27:379–423 and 623–656, 1948.

[25] D. Slepian and J. Wolf. Noiseless coding of correlated information sources. *IEEE Transactions on Information Theory*, 19:471–480, 1973.

[26] J. Watkinson. *New Protocols for Asymmetric Communication Channels.* Masters Thesis (in preparation), University of Toronto, 2000.

[27] A.C. Yao. Some complexity questions related to distributive computing. In 11^{th} *ACM Symposium on Theory of Computing*, pages 209–213, 1979.