Protocols for Asymmetric Communication Channels

Micah Adler* Bruce M. Maggs

December 1997
CMU-CS-97-191

School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

* Department of Computer Science
University of Toronto
10 King's College Road
Toronto ON, M5S3G4
Canada

Micah Adler is supported by an operating grant from the Natural Sciences and Engineering Research Council of Canada, and by ITRC, an Ontario Centre of Excellence. This research was conducted in part while he was at the Heinz Nixdorf Institute Graduate College, Paderborn, Germany. Bruce Maggs is supported in part by the Air Force Materiel Command (AFMC) and ARPA under Contract F196828-93-C-0193, by ARPA Contracts F33615-93-1-1330 and 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".

Keywords: asymmetric communication, communication protocol

Abstract

This paper examines the problem of communicating an n-bit data item from a client to a server, where the data is drawn from a distribution D that is known to the server but not to the client. Since this question is motivated by asymmetric communication channels, our primary goal is to limit the number of bits transmitted by the client. We present several protocols in which the expected number of bits transmitted by the server and client are $O(n)$ and $O(H(D))$, respectively, where $H(D)$ is the entropy of D, and can thus be significantly smaller than n. Shannon's Theorem implies that these protocols are 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 protocol provides a tradeoff between the computational effort and the number of rounds. These protocols are complemented by several lower bounds and impossibility results. We show 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 $n-1$. 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, for an arbitrary distribution D, the minimum expected total number of bits sent by the server and client. Furthermore, the problem remains undecidable even if only an approximate solution is required, for any reasonable degree of approximation.

1 Introduction

In the summer of 1995, the second author set out to establish a high-speed wireless internet connection between the Carnegie Mellon campus and his home approximately one mile away. A directional antenna was mounted above the tallest tower on campus, and a matching antenna was installed at home. The antennas were driven by WaveLAN transceivers, which implement 2 megabit/second wireless ethernet. The installation was successful, but within a few months the performance of the wireless connection deteriorated to the point that it was no longer usable. This problem coincided with the deployment of a campus-wide wireless network intended to provide laptop users with uninterrupted access as they roamed the campus [6]. Unfortunately, this network used the same WaveLAN technology and carrier frequency, and as a result, packets traveling from home to campus were often lost in transit.

Ultimately, the goal of establishing a high-speed bidirectional wireless connection was abandoned, and packets were instead routed from home to campus across an ordinary telephone line using a 28.8 kilobit/sec modem at each end. The resulting connection was highly asymmetric, with download speed exceeding upload speed by a factor of 69-to-1. The connection proved adequate, however, to support an X-terminal and web browser at home, and has been in daily use for several years. Some tasks, however, are limited by the slow upload speed.

In the past two years a number of commercial asymmetric networking technologies have also been introduced. For example, using ordinary telephone lines, 56k modems can download at up to 56kbs, but can upload data at a maximum rate of 33.6kbs. In some cities, telephone companies are now offering much more asymmetric network connections. For example, in Pittsburgh trials of asymmetric digital subscriber loops (ADSL) have begun. These ADSLs provide a download speed of 1.5mbs, and an upload speed of 64kbs. As another example, The DirecPC network connection offered by Hughes beams data down from a satellite 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. Independent of whether the underlying medium is asymmetric, it has been observed that home users typically download much more data than they upload.

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 data item 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

We study an asynchronous model based on Yao's two-party communication complexity model [14]. In order for the client to transmit its n-bit data item to the server, the client and the server communicate bits to each other, as specified by some fixed protocol $\mathcal{P}$. At each step of the protocol, $\mathcal{P}$ specifies 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. When the protocol terminates, the server must have enough information to determine with certainty the n-bit data item.

It is already well known, and not difficult to prove, that if the server has no information about the n-bit data item held by the client, then in the worst case the number of bits sent by the client must be at least n. This information-theoretic lower bound would seem to imply that there is no way to exploit the high-speed downlink. There are many circumstances, however, in which the server has some information about the data item held by the client. For example, if the client is sending a sequence of keystrokes to the server, the server may know the frequency with which the client presses any particular key. Throughout this paper we assume that the n-bit data item held by the client has been drawn randomly from a probability distribution, and that this distribution is known to the server.

In the keystrokes example, it is reasonable to assume that both the client and the server know the distribution, since both have seen a history of keystrokes made by the client. In this case, the client and the server can agree on a data compression protocol for the client to use in encoding its data. For example, suppose that the client uses a static Huffman-coding scheme [7]. Then the data can be transferred in one round with no bits sent by the server and with at most $H(D)+1$ expected bits sent by the client, where $H(D)$ is the *entropy* of the distribution D, a quantity that varies between 0 and n, and is given by the equation

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

where x_i is an n-bit string and p_i is the probability of x_i. Notice that because both the client and the server know the distribution, the high-speed downlink is not utilized in this example.

But what if only the server knows the distribution? Throughout this paper, we assume that the client does not know the distribution. Although this seems counterintuitive at first, there are natural situations consistent with this assumption. As an example, suppose that the client has filled out a form on an internet web page, and is to send its response to a server. The server, which has seen many replies, may know the distribution, but the client is unlikely to. As another example, suppose that each client is a probe that is designed to take an experimental sample and report it back to the server. The probe will not have access to the samples taken by other probes, and may not be able to make a long transmission back to the server.

1.2 Our results

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, ρ is the expected number of rounds (defined below), and λ is the expected computational effort expended by the server (also defined below). These parameters are functions of n, the number of bits in the data item, and $H(D)$, the entropy of the distribution D.

A round of the protocol is defined to be a maximal set of consecutive bits sent by the server (without any bits sent in between by the client), followed by a maximal set of consecutive bits sent by the client. All the bits sent by the server (or client) in a round can be transmitted without waiting for a response from the client (or server, respectively), and thus minimizing the number of rounds required by a protocol is an important consideration in many scenarios. For example, this is the case when the time required for a round-trip communication is large and does not depend on the number of bits communicated.

The computational effort λ expended by the server is quantified as follows. We assume that the server has access to the distribution on the string held by the client via a black box that answers queries of the form "What is the cumulative probability of 4-bit data items matching the pattern `0*11`?" In this example n is 4, and the server is asking for the sum of the probabilities of the strings `0011` and `0111`. The parameter λ is simply the number of such queries made by the server. In all of our protocols, the additional computation time required of the server in the random-access machine (RAM) model [1] is at most $O(\lambda \log \lambda)$. The computation time required by the client is at most $O(n)$.

We require that upon the termination of a protocol, the server knows the n-bit data item held by the client with certainty, i.e., there is no probability that the server incorrectly identifies the data item. The number of bits transmitted by both parties, the number of rounds, and the computational effort expended by the server, however, may all be random variables taken over the distribution of data items and over random choices made by the algorithms underlying the protocols.

We begin by describing a $(3n, 1.71H(D)+1, 3n, 1.71H(D)+1)$ protocol. We call this protocol **Computationally-efficient** because the expected number of black-box queries performed by the server is asymptotically optimal.

Next, we present an $(O(n), O(H(D)+1), 2^n, O(1))$ protocol. This protocol is called **Round-efficient**, because the expected number of rounds required is only a constant, which is optimal.

Our third protocol, **Computation-Rounds-Tradeoff**(c), allows us to achieve 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) is an $(O(n), O(H(D)+1), O(\frac{n2^c}{c}), O(\frac{n}{c}))$ protocol.

It is worth noting that the actual speeds of the downlink and uplink do not appear in our analyses of these protocols. Although these protocols were motivated by networks with asymmetric transfer speeds, in fact they can be applied in any situation in which it is desirable to reduce the number of bits transmitted by the client, provided that the server knows the distribution, but the client does not.

We complement these upper bounds by proving a number of impossibility results and lower bounds.

We begin by observing that all three of our protocols are asymptotically optimal in the number of bits sent by the client, $O(H(D))$. Next, we show that they are also all existentially asymptotically optimal in the number of bits sent by the client and the server together, $O(n)$. By existentially, we mean that for any h, there are distributions with entropy $H(D) = h$ for which the expected total number of bits sent by both parties must be $\Omega(n)$.

We also 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. This follows from a proof that it is undecidable to compute, for an arbitrary distribution D, the value $OPT(D)$, the minimum expected total number of bits that the server and client must exchange in order to solve the problem. Furthermore, the problem remains undecidable even if only an approximate solution is required. For example, computing a value that is guaranteed to be between the inverse of Ackerman's function applied to $OPT(D)$ and Ackerman's function applied to $OPT(D)$ is undecidable.

We conclude by showing that the 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 sum of the number of bits sent by the client plus the number of black-box queries is at least n. We also show that for any single-round protocol in which the client sends $O(H(D))$ bits, there are distributions for which the server must send an exponential number of bits, $\Omega(n2^{H(D)})$.

1.3 Previous work and related work

There is a wealth of literature on two-party communication complexity. Most of this work, however, examines symmetric communication channels, and analyzes the total number of bits transmitted by the two parties, and sometimes the number of rounds. A good reference is the recent book by Kushilevitz and Nisan [8].

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 [11, 12]. The last paper is most closely related to this one. It presents 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. It then applies these techniques to a several fundamental problems. As an example, one of the problems it considers is the *membership* problem. In this problem the server holds a set S of strings, and the client holds a single string x. The goal of both parties is to determine if x belongs to S. The paper also examines generalizations of the membership problem such as the *disjointness* problem. In this problem the server and client hold sets S and C, and the goal is to determine if the sets are disjoint. Other problems include the *span* problem, in which the server holds the basis of a vector space, the client holds a vector, and the goal is to determine of the client's vector lies in the server's space, and the *greater than* problem, in which the server and client each hold an integer, and the goal is to determine which integer is larger.

A recent study has 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 [2]. 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 Upper Bounds

In this section we provide three protocols. All three are within a constant fraction of optimal in terms of the number of bits sent by the client. They are also all existentially optimal in the number of bits sent by the server. The first is 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 the number of rounds required. In the following, let D represent the distribution known to the server, and let $D(x)$ be the probability assigned to the string x by the distribution D.

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 this information allows 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, p_Q can be adjusted to reflect that the actual string cannot be in the set X by first subtracting the weight of all strings in X that are consistent with Q, and then dividing the result by the weight of all the strings not in X. We call the resulting value the *exclusion adjusted* probability.

The protocol is defined as follows:

Repeat the following until the entire string is known:

- 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 that occurs with probability between $\frac{1}{3}$ and $\frac{2}{3}$, or that extends to the end of the string.
    - Query the black box for s0 occurring starting in the first unknown bit position.
    - If the exclusion adjusted probability of the value returned by the black box is $> \frac{2}{3}$, then a 0 is appended to 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.
- The server sends this 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".

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 $\frac{p}{2}$.

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 $O(H(D) + 1)$ bits. For any input distribution D, we model the bits sent by the client as a tree, where 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". 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 gives us the following important fact. 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 $\sum_{x_i} D(x_i)(1 + \log_{2/3} D(x_i)) = 1 + H(D)/\log(\frac{3}{2}) \approx 1 + 1.71H(D)$.

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 possible matches sent by the server for the i^{th} bit 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.

Protocol **Round-efficient**

For any distribution D, let $T(D)$ represent the strings in sorted order from most likely to occur to least likely to occur. Let $\tau(D)$ represent 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, or contains between $\frac{1}{3}$ and $\frac{2}{3}$ of the remaining probability weight.

We can compare the partition of the strings into the sets $\mathcal{X}_i$ with the construction of a Fano code [5] (see also [4]). To construct a Fano code, the strings are likewise sorted in order of probability, and then divided into as close to two equally probable sets as possible. The first bit of the codeword is assigned to a 1 if the string lies in the first set, and a 0 if the string lies in the second set. However, for a Fano code, this same process is repeated on both sets as many times as is possible. In our construction, we only subdivide the set which contains the less likely strings. Instead of subdividing the other set, we reduce the number of rounds required by using hashing to differentiate between strings in that set. The difficult portion of this technique is to demonstrate that the client can use hashing in a manner that does not increase the number of bits sent by more than a constant factor.

We here use $\mathcal{F}_n$, 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 integers chosen uniformly and independently at random from the range $[0 \ldots 2^n - 1]$, 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 [10]).

- The server queries the black box to find $D(x_i)$ for all possible strings x_i, and uses this information to determine the partition $\tau(D)$. To do this, the server sorts the strings based on $D(x_i)$.

- The server sends to the client a randomly chosen hash function $F \in \mathcal{F}_n$.
- Let $i = 1$ and let $h = 0$.
- Repeat the following until x, the client's string, is known by the server.
  - The server sends to the client the binary representation of $h' = \lceil \log h_i \rceil$.
  - If $h' > h$, the client sends to the server bits $h+1$ through h' of $F(x)$. Note that this is sufficient for the server to know the first h' bits of $F(x)$.
  - $h = \max(h, h')$.
  - The server finds all strings $x' \in \mathcal{X}_i$ such that the first h bits of $F(x')$ are the same as the first h 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 $O(1 + H(D))$. 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 $\bar{\tau}$.

We describe the code $\bar{\tau}$ as a tree. In this tree, 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 $0 \le k \le r - 2$ 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 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 [5].

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 $E(\bar{\tau}) = O(1+H(D))$. As before, along any path from the root to a leaf, there is at most one node that is not balanced. 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 $\sum_{x_i} D(x_i)(1 + \log_{2/3} D(x_i)) = 1 + H(D)/\log(\frac{3}{2}) \approx 1 + 1.71H(D)$.

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 $E(\bar{\tau})$. We assume that there is more than 1 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 $h_j = 1$. When $h_j > 1$, let x_m be a minimum-depth leaf in $\mathcal{X}_j$ such that if x'_m, the other child of the parent of x_m, is also a leaf, then $D(x_m) \geq D(x'_m)$. Let r_j be the root of the subtree defined 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, and, since we have a pairwise independent hash function, the expected number of strings in the list is ≤ 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 need 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 $h_i > 1$, $q_{i+1} \geq \frac{s_i}{9}$.
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, $s_{i+1} \geq \frac{s_i}{3}$, which combined with Fact 1, 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 imply 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 need to 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, the string held by the client. For any set $\mathcal{H}_j$, the number of bits needed to represent $\lceil \log h_j \rceil$ is $\log\log h_j + o(\log h_j) < \log n + o(\log n)$. The number of bits needed to describe the hash function is $2n$. Since we have a pairwise independent hash function, for each set $\mathcal{X}_i$ that is examined the expected number of strings that map to the same image as x_i, not counting x_i itself, is ≤ 1. 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 $6n + o(n)$. ■

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 $(\frac{7}{2}n, 2H(D)+O(1), 2^n, 3)$ protocol. 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 $(\frac{3}{2}n, 2H(D) + O(1), 2^n, 3)$ protocol.

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

Protocol **Computation-Rounds-Tradeoff(c)**

For c a positive integer between 1 and n, repeat the following until the entire string is known:

- 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 the value of the bits determined 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 a* $(O(n), O(H(D)+1), O(\frac{n2^c}{c}), O(\frac{n}{c}))$ *protocol.*

Proof: We first show that the expected number of bits sent by the client is $O(H(D) + 1)$. The possible "y" and "n" bits sent by the client define a tree ν, as before. 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** that can be made over all possible strings held by the client. 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.

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 in 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 $\bar{\nu}$, there can never be two consecutive unbalanced nodes, with the possible exception of the last two nodes. 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 $O(\frac{n2^c}{c})$.

The bound on the number of rounds required follows from the fact that 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 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 Bits Sent by The Client and The Server

Theorem 4, below, implies that our protocols, for all of which the expected number of bits sent by the client is $O(H(D))$, are optimal in terms of this measure.

Theorem 4 (Shannon [13]) *For any distribution D, the expected number of bits sent by the client is at least $H(D)$, where $H(D)$ is Shannon's entropy of the distribution D.*

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 need to be sent by the server. To do this, we show that when the distribution is chosen from a broad class of distributions, then the expected total number of bits that need to 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 $O(H(D))$ bits. We demonstrate in Section 3.1 that there are distributions where the total number of bits sent can actually be much less than n. However, we also demonstrate that designing a protocol that uses close to the minimum total number of bits for all distributions is not possible.

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

Definition 2 *A multi-set of distributions is* symmetric *if the distributions in the set can be partitioned into subsets such that within each subset, (1) each distribution appears only once, and (2) for any distribution D_1 in the subset and for any permutation π of the 2^n strings x_i, there is a distribution D_2 in the same subset, such that for all x_i, $D_1(x_i) = D_2(\pi(x_i))$. We call this partition the* balancing partition.

Intuitively, a multi-set of distributions is symmetric if no preference is given to any specific string.

Theorem 5 *For any protocol and for any distribution chosen uniformly at random from any set of onto distributions that is symmetric, the expected total number of bits sent by the client and the server is at least $n - 1$.*

Proof: We prove the Theorem for the case where the balancing partition consists of a single subset. When the balancing partition consists of many subsets, choosing a distribution is equivalent to first choosing a subset in the balancing partition, and then choosing a distribution from within that subset. Thus, the result for a set with only a single subset in the balancing partition implies the more general result stated in the Theorem.

Any set with only a single subset in the balancing partition can be described by a sequence of pairs $PN = (p_1, N_1), (p_2, N_2), \ldots, (p_k, N_k)$, where $p_i > p_j$ for $i < j$ and where for each distribution in the set, there are exactly N_i strings that occur with probability p_i. Let x_l be the string given initially to the client, and let D_m be the distribution given initially to the server. Here, l represents the index of the string held by the client, and m represents the index of the distribution held by the server. Let I be the unique value such that $p_I = D_m(x_l)$. We here prove a lower bound for the problem where both the client and the server know the sequence PN at the start of the protocol, and the server must determine not only the string x_l, but also the value I. Since this problem requires no more communication than the original problem, a lower bound for this problem also applies to the original problem.

Let E_i be the expected total number of bits sent by the client and the server, conditioning on $I = i$, where this expectation is taken over both the random choice of D_m and the choice of x_l using the chosen distribution D_m. We prove that for any fixed i, $1 \leq i \leq k$, $E_i \geq n-1$. Since the actual value of I computed by the server is a distribution over $1 \leq i \leq k$, this suffices to prove the Theorem. Note that for every D_m, the client must send a different set of bits to the server for every string x_l. Also, all of the N_i strings where $I = i$ occur with equal probability, and thus when $N_i \geq \frac{2^n}{2}$, $E_i \geq n-1$.

To show that $E_i \geq n-1$ when $N_i < \frac{2^n}{2}$ we view the communication task of the server determining I as a communication matrix, where the rows of the matrix represent the input given to the client (i.e., there are 2^n rows), and the columns represent the distribution given to the server (i.e., there are $\binom{2^n}{N_1; N_2; \ldots N_k}$ columns). The entry of the matrix in row l and column m contains the value of I when the server starts with distribution D_m and the client starts with string x_l. We say that an input to such a problem is the pair (l, m).

We use a technique based on the idea of monochromatic rectangles, a common technique in communication complexity developed in [14]. This technique uses the fact that any communication protocol partitions the communication matrix into *rectangles*, each consisting of the matrix entries in the intersection of a specific subset of the rows of the communication matrix with a specific subset of the columns of the matrix. The transcript of bits communicated by the client and the server is the same for any two inputs that represent two matrix entries within the same rectangle, and different for any two inputs that represent matrix entries in different rectangles. For a proof of this, see for example [8].

Several classical results in communication complexity show that there must be a large number of rectangles by showing that each rectangle must be monochromatic: all entries in the rectangle must contain the same value. This is required since typically both the client and the server are required to know the result at the end of the protocol. However, in the problem we consider, the color of a matrix entry is the I value in that entry, and only the server is required to know I at the end of the protocol. Thus, for the problem discussed here, within each rectangle, every **column** must be monochromatic. We call such a rectangle *column monochromatic*. Note that only rectangles associated with distributions that are onto are required to be column monochromatic. If there were an input pair that is guaranteed to not occur, the server is not required to differentiate this input from one that does occur, and thus the input that does not occur could be in the same column of the same rectangle as an input that does occur.

We provide an upper bound on the number of times the value i can appear in a single rectangle. Consider any rectangle that consists of r rows. Let C_i be the set of columns of that rectangle that contain the value i, and let the r rows be denoted by R. For this rectangle to be column monochromatic, for each string x_l that appears in R, and for each $D_m \in C_i$, $D_m(x_l) = p_i$. This means that $r \leq N_i$. By counting the number of ways that the probability of the strings x_l not in R can be set, this also implies that $|C_i| \leq \binom{2^n-r}{N_1;\ldots N_i-r;\ldots N_k}$. Thus, the maximum number of times that i can appear in a single rectangle is

$$\max_{r \leq N_i}\left[r \cdot \binom{2^n - r}{N_1;\ldots N_i - r;\ldots N_k}\right].$$

When $N_i < \frac{2^n}{2}$, this is maximized when $r = 1$. This implies that the number of i's in a single rectangle is at most $\binom{2^n-1}{N_1;\ldots N_i-1;\ldots N_k}$. For z a rectangle containing the value i, let q_z be the probability that the input pair (x_l, D_m) is in rectangle z, conditioned on $I = i$. Since every input pair that results in $I = i$ occurs with the same probability, when $N_i < \frac{2^n}{2}$, $q_z \leq \frac{1}{2^n}$, $\forall z$. The value $\frac{1}{2^n}$ is obtained by dividing the maximum number of times that i can appear in any one rectangle of the communication matrix by the total number of times that it appears.

Let $\mathcal{E}_i$ be the entropy of the transcript of bits sent by the client and the server. The fact that the transcript of bits sent by the client and the server is different for every rectangle together with the upper bound on q_z imply that for any i such that $N_i < \frac{2^n}{2}$, $\mathcal{E}_i \geq n$. This in turn implies that $E_i \geq n$. For any i such that $N_i \geq \frac{2^n}{2}$, $E_i \geq n-1$, and thus the *a priori* expected number of bits communicated is at least $n-1$. ■

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. Consider for example the following distribution, D_s, described by giving a technique of choosing strings from the distribution. The last $\log n$ bits of the n bit string x_i are chosen by independent flips of a fair coin. Let t be the value of the binary number represented by these bits. When $t < n - \log n$, the remainder of the bits in the string are set to 0, except the t^{th} bit, which is set to 1. When $t \geq n - \log n$ all the remaining bits are set to 0.

For the distribution D_s, the entire string is determined by the last $\log n$ bits. However, protocol **Computation-efficient** sends possible matches for prefixes of the string, and in order to find a prefix that occurs with probability between $\frac{1}{3}$ and $\frac{2}{3}$, the server has to send a prefix to the client that has length $\lceil \frac{n}{3} \rceil$. For distribution D_s, we would be much better off with a protocol that specifies an order on the bits that the server is trying to match. This could be done for D_s by having the server send $\log^2 n$ bits indicating which $\log n$ bit positions to consider first.

Unfortunately, it is not sufficient for the server to simply specify the order in which the bits should be examined. Consider for example the distribution $D_{s'}$, where the string consists of $n-1$ 0s and a single 1, placed uniformly at random. Here, regardless of what order the server attempts to match bits held by the client, the number of bits that must be matched in order to find a successful match with probability between $\frac{1}{3}$ and $\frac{2}{3}$, is $\frac{n}{3}$. However, this distribution has a very short description ($O(\log n)$ bits) that could be sent to the client. Once the client knows what the distribution is, it can simply send the server $\log n$ bits describing where the 1 is in her string.

Let $OPT(D)$ represent the minimum expected total number of bits sent when the client has a sample drawn from the distribution D, which is known only to the server. The distributions D_s and $D_{s'}$ serve to demonstrate that distributions do exist where 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. However, we next show that it is not possible for any protocol to use $OPT(D)$ bits for every distribution D. In fact, we show that any function that provides a non-trivial approximation to $OPT(D)$ 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, no other protocol could guarantee to be optimal for all distributions.

The proof of the following Theorem uses Kolmogorov complexity, and is in fact motivated by Kolmogorov's proof (see for example [9]) that the Kolmogorov complexity of a string is not a recursive function. Recall that the Kolmogorov complexity of a string x, which we here denote $\mathcal{K}(x)$, is the minimum description length of the string x. We here use the definition that $\mathcal{K}(x)$ is 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.

Theorem 6 *Let $f(D)$ be any function from distributions D to $\Re$ such that*

$f(D) \leq OPT(D)$ and as $OPT(D) \to \infty$, it is also the case that $f(D) \to \infty$. Such a function $f(D)$ is not recursive.

Proof: Let $f(D)$ be any such function. We assume that $f(D)$ is recursive and reach a contradiction. We assign a distribution D_i for each natural number i, where D_i is the distribution over $n = \lfloor \log i \lfloor$-bit strings where the string corresponding to the binary representation of $j = i - 2^{\lfloor \log i \rfloor}$ occurs with probability $1 - 2^{-2\lfloor \log i \rfloor}$, and all other strings occur with equal probability.

Using these 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 $f(D_i) \geq m$. Let B_n be the set of all 2^n distributions D_i over n-bit strings. The set B_n is symmetric, and each distribution D_i is onto, and thus Theorem 5 implies that some distribution $D_i \in B_n$ is such that $OPT(D_i) \geq n - 1$. This means that the value of $OPT(D_i)$ takes on arbitrarily large values as i increases. Since $f(D_i)$ increases without bound as $OPT(D_i)$ increases, we see that $F(m)$ is well defined for each natural number m.

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

Proof (of Claim): By our construction, $OPT(D_{F(m)}) \geq m$, so it suffices to show that for all i, $\mathcal{K}(i) \geq OPT(D_i) - c$ for some constant c. This follows from the fact that $OPT(D_i)$ can be at most an additive constant larger than 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 1 followed by the description of the string i of length $\mathcal{K}(i)$. The client responds to the server with a 1 if it indeed has the string $j = i - 2^{\lfloor \log i \rfloor}$ and a 0 followed by the actual value of the string otherwise. When the distribution received by the server is not one of the D_i, the server sends the client a 0, and this is followed by any other protocol. Note that in such a protocol, when the server has a distribution D_i, the expected total number of bits used is at most $\mathcal{K}(i) + 3$. ■

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. ■

Consider for example any approximation function for $OPT(D)$ that is guaranteed to return a value $g(D)$ such that $\alpha^{-1}(OPT(D)) \leq g(D) \leq \alpha(OPT(D))$, where α and α^{-1} are Ackerman's function and its inverse, respectively. The function $f(D) = \alpha^{-1}(g(D)) \leq OPT(D)$, and $f(D)$ grows without bound as $OPT(D)$ grows, albeit very slowly. Thus, $f(D)$ is not recursive, which in turn implies that $g(D)$ is also not recursive.

4 Lower Bounds on Computation and Rounds

We show that 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 7 *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, for any black box query, the response 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 bits (leaving $n-k$ bits as wild cards), then the two possible answers are $2^{n-k} \cdot \frac{1-p}{2^n-1}$, (in the case where the single likely string does not match the query), and $(2^{n-k}-1) \cdot \frac{1-p}{2^n-1} + p$, (in the case where the single likely string does match the query).

The actions of the server can be viewed as a 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. Each of the 2^n possible strings held by the client must result in reaching a different leaf, and thus the average height of the leaves must be at least n. ■

Protocol **Round-efficient** is trivially within a constant factor of the best possible in terms of the expected number of rounds required. We show here that a protocol that always completes in a single round would require either the number of bits sent by the client to be much larger than the minimum, or the number of bits sent by the server to be much larger than the minimum. A *single-round protocol* is defined as 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.

Theorem 8 *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 $c \cdot H \leq \frac{n}{16}$. 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 $\frac{n2^H}{4}$.*

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 $\binom{2^n}{2^H}$ distributions, one for each subset of 2^H 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 $1-2^{-n}$, 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.

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, on average the server must send $\log\binom{M}{m}$ bits to the client. Thus, the following lemma directly implies the Theorem.

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 x, and the expected number of bits the client sends to the server is at most cH, implies a protocol B for the subset identification problem with $M = 2^n$ and $m = 2^H$, where, on average, the server sends $x + \frac{3}{4}\log\binom{M}{m}$ bits to the client.*

Proof (of Lemma): The protocol B proceeds as follows. The server examines the $M = 2^n$-bit input string I and determines the $m = 2^H$ 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 $4cH$ 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^{4cH}. Furthermore, the number of bits that are 1's in I that are not 1's in I' is at most $\frac{2^H}{2}$.*

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 $4cH$ bits is at most 2^{4cH}. 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 $\frac{1}{1-2^{-n}} cH$. Thus, by Markov's inequality, the fraction of likely strings where the client sends more than $4cH$ bits is $< \frac{1}{2}$. ■

The server only needs 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 $\frac{2^H}{2} \log 2^n$ bits, and the latter requires at most $2^H \log 2^{4cH}$ bits. Using the fact that $4cH \leq \frac{n}{4}$, this is $\leq \frac{3}{4} \log \binom{M}{m}$ bits, and the lemma follows. ■ ■

References

[1] A. Aho, J. Hopcroft, and J. Ullman. *The Design and Analysis of Computer Algorithms.* Addison-Wesley, Reading, Massachusetts, 1974.

[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. To appear.

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

[4] T. Cover and J. Thomas. *Elements of Information Theory.* Wiley and Sons, New York, 1991.

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

[6] A. Hills and D. B. Johnson. A wireless data network infrastructure at carnegie mellon university. *IEEE Personal Communications*, 3(1):56–63, February 1996.

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

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

[9] 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.

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

[11] 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.

[12] 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.

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

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