Dynamic Asymmetric Communication

STUDENT PAPER
Travis Gagie

Department of Computer Science
University of Toronto
travis@cs.toronto.edu

Abstract. In Adler and Maggs' asymmetric communication problem, a server with high bandwidth tries to help clients with low bandwidth send it messages. We give four new asymmetric communication protocols and show they are robust with respect to changes in the messages' distribution. Three of our protocols require only one round of communication for each message.

1 Introduction

Internet users usually download more than they upload, and many technologies have asymmetric bandwidth — greater from servers to clients than from clients to servers. Adler and Maggs [1] considered whether a server can use its greater bandwidth to help clients send it messages. They proved it can, assuming it knows the messages' distribution. We argue that assumption is often unwarranted and, fortunately, unnecessary.

Suppose a number of clients want to send messages to a server; each client only knows its own messages but, at any point, the server knows all the messages it has already received. Adler and Maggs assumed the server, after receiving a sample of messages, can accurately estimate the distribution of all the messages. They proved it can use that knowledge to reduce the average number of bits each of the remaining clients sends, to roughly the entropy of the distribution. Their work has been improved and extended by several authors, whose results are summarized in Table 1, and used in MIT's Infranet anti-censorship project [5,6]. However, while implementing Infranet, Wang [17] found the messages' distribution changed over time — the sample was unreliable.

In this paper we present and analyze four asymmetric communication protocols that deal with changing distributions using techniques from data compression; our results are summarized in Table 2. In Section 2 we give a dynamic version of Watkinson, Fich and Adler's Bit-Efficient Split protocol [18]. We present and analyze our TreeQuery and ListQuery protocols in Section 3. For the former, we reduce the number of rounds to 1 at the cost of increasing the number of bits the server sends and then, for the latter, we reduce the number the server sends at the cost of increasing the number the clients send. Finally, in Section 4 we show how Bentley, Sleator, Tarjan and Wei's Move-to-Front compression algorithm [2] can be turned into an elegant asymmetric communication protocol, which we call Move-to-Front-and-Truncate. Our protocols can be implemented so that each party's computation is proportional to the number of bits it sends and receives.

References	Bits sent by Server	Bits sent by Client	Rounds
[1,13]	$3(\lfloor \log N \rfloor + 1)$	$1.09H + 1$	$1.09H + 1$
[1]	$O(\log N)$	$O(H + 1)$	$O(1)$
[18]	$(H + 2)(\lfloor \log N \rfloor + 1)$	$H + 2$	$H + 2$
[18]	$O(2^k H \log N)$	$H + 2$	$(H + 1)/k + 2$
[10]	$kH(\lfloor \log N \rfloor + 1) + 1$	$H \log_{k-1} k + 1$	$H/\log k + 1$
[3]	$(k + 2)(\lfloor \log N \rfloor + 1)$	$\frac{H \log(k+2)}{\log(k+2)-1} + \log(k + 2)$	$\frac{H}{\log(k+2)-1} + 1$

Table 1. Suppose a server tries to help a client send it one of N messages, chosen according to a distribution with entropy H that is known to the server but not the client. Adler and Maggs [1], Watkinson, Adler and Fich [18], Ghazizadeh, Ghodsi and Saberi [10] and Bose, Krizanc, Langerman and Morin [3] gave protocols for this problem, whose expected-case upper bounds appear above; the last three protocols take a parameter $k \geq 1$. This table is based on one given by Bose *et al.*; we use some new notation.

Algorithm	Bits sent by Server	Bits sent by Client	Rounds
DBES	$(H + O(1)) \log N$	$H + \frac{n \log N}{m} + O(1)$	$H + O(1)$
TreeQuery	$2N - 1$	$H + \frac{n \log N}{m} + O(1)$	1
ListQuery	$\lfloor N^{1/k} \rfloor (\lfloor \log N \rfloor + 1)$	$kH + \frac{n \log N}{m} + O(1)$	1
MFT	$\lfloor N^{1/k} \rfloor (\lfloor \log N \rfloor + 1)$	$kH + \frac{n \log N}{m} + O(1)$	1

Table 2. Suppose a server tries to help clients send it m messages whose distribution — known to neither the server nor the clients — has entropy H; of N possible distinct messages, n occur. We present four protocols for this problem and prove upper bounds on the average cost of sending each message, which appear above; the last two protocols take a parameter $k > 1$.

2 Dynamic Bit-Efficient Split

Let $S = s_1, \ldots, s_m$ be a sequence of messages some clients want to send a server. Let N be the number of possible distinct messages and let n be the number that occur in S. Let $H = \sum_{a \in S} \frac{\#_a(S)}{m} \log \frac{m}{\#_a(S)}$, where $a \in S$ means message a occurs in S, $\#_a(S)$ is a's frequency in S and log means $\log_2$; that is, H is the entropy of the messages' distribution, also called the 0*th-order empirical entropy of S*.

For the moment, assume the server knows the messages' distribution beforehand. For Watkinson, Adler and Fich's Bit-Efficient Split protocol [18], the server uses a linear-time algorithm due to Mehlhorn [14] to build a leaf-oriented binary search tree T on the distinct messages in S; the jth leaf of T stores the jth lexicographically largest message $a \in S$, at depth at most $\left\lceil \log \frac{m}{\#_a(S)} \right\rceil + 1$; each internal node has exactly two children (i.e., T is *strictly* binary) and stores the lexicographically largest message in its left subtree.

For each message s_i, the server starts at the root of T and descends to the leaf v of T storing s_i. At each proper ancestor u of v, the server sends the client the message a stored at u; if $a \geq u$, then the client responds with 0 and the server descends to u's left child; if $a > u$, then the client responds with 1 and the server descends to u's right child.

Without loss of generality, assume messages are $\lfloor \log N \rfloor + 1$ bits long; otherwise, we use their indices in lexicographic order. Straightforward calculation shows that, on average, each message takes fewer than $H + 2$ rounds, during which the server sends fewer than $(H+2)(\lfloor \log N \rfloor + 1)$ bits and the client sends fewer than $H + 2$ bits.

Dynamic Bit-Efficient Split (DBES) does not require the server to know the distribution beforehand. Next, we give a simple but slow implementation of DBES; for each message, the server sends $(H + O(1)) \log N$ bits, on average, but performs $O(N)$ computations. It is possible to reduce the number of computations the server makes to $O((H+1) \log N)$, using a data structure developed for dynamic alphabetic coding [9] and sorting low-entropy sequences [8].

For each message s_i, the server builds a leaf-oriented binary search tree T_i on all possible distinct messages; the jth leaf of T_i stores the jth lexicographically largest possible message a, at depth at most $\left\lceil \log \frac{i}{\#_a(s_1,\ldots,s_{i-1}) + 1/N} \right\rceil + 1$. (Notice $\sum_a \#_a(s_1, \ldots, s_{i-1}) = i - 1$, so $\sum_a \big(\#_a(s_1, \ldots, s_{i-1}) + 1/N\big) = i$.)

The server starts at the root of T_i and descends to the leaf v storing s_i. If v is high in the tree, the server descends as in Bit-Efficient Split; however, once the server has descended below depth $\lceil \log i \rceil + 1$, it knows the client's message must be one it has not seen before. In the latter case, the server signals the client by sending the same message twice (notice it never does this for Bit-Efficient Split); the client responds with s_i.

Our analysis relies on the following technical lemma (see [9]), a proof of which appears as an appendix.

Lemma 1.

$$\sum_{i=1}^{m} \log \frac{i}{\max\big(\#_{s_i}(s_1, \ldots, s_{i-1}), 1\big)} \leq (H + O(1))m \ .$$

Using Lemma 1, it is easy to bound the number of bits the server sends, the number the clients send and the number of rounds in DBES.

Theorem 1. *Suppose some clients send S to a server using DBES. On average, each message takes $H + O(1)$ rounds, during which the server sends $(H + O(1)) \log N$ bits and the client sends $H + \frac{n \log N}{m} + O(1)$ bits.*

Proof. There is one round for each level the server descends in a tree. For each message s_i, the server descends at most

$$\begin{aligned} &\max\left(\left\lceil \log \frac{i}{\#_{s_i}(s_1, \ldots, s_{i-1})} \right\rceil + 1, \lceil \log i \rceil + 2\right) \\ &< \log \frac{i}{\max\big(\#_{s_i}(s_1, \ldots, s_{i-1}), 1\big)} + 2 \end{aligned}$$

times, by Lemma 1, there are $H+O(1)$ rounds. The server sends $\lfloor \log N \rfloor + 1$ bits during each round. If s_i has occurred before, then the client sends 1 bit during each round; otherwise, it sends 1 bit during each except the last, when it may send $\lfloor \log N \rfloor + 1$ bits. □

As an aside, we note DBES can easily be modified so the trees are only based on the distribution of recent messages — a data compression technique for increasing robustness. The server maintains a queue of messages; for each message s_i, it builds a tree based on the distribution of messages in the queue; after receiving s_i, it dequeues one message and enqueues s_i. Knuth [12] discusses "sliding windows" such as this.

3 TreeQuery and ListQuery

Our next protocol, *TreeQuery*, is a simple modification of DBES. Instead of querying each client repeatedly to find a path in a tree, the server encodes and sends the whole tree; the client finds the path and sends back all of what would have been its responses. To encode the tree, the server performs a preorder traversal, recording each internal node as a 1 and each leaf as a 0. Since Mehlhorn's algorithm is linear, both the server and the client perform $O(N)$ computations.

Theorem 2. *Suppose some clients send S to a server using TreeQuery. For each message, the server sends $2N-1$ bits and, on average, the client sends $H + \frac{n \log N}{m} + O(1)$ bits.*

Proof. For each message, the server encodes and sends a strict binary tree on N leaves, which takes $2N-1$ bits; the number of bits the clients send is bounded as in Theorem 1. □

As an aside, we note we can modify TreeQuery to slightly reduce the average number of bits a client sends, at the cost of possibly again increasing the number of bits the server sends. The server maintains a dynamic Huffman tree [11] and, for each round, encodes and sends the tree's current state; the client responds with the codeword for its message; the server updates the Huffman tree as it would in dynamic Huffman coding [12,16] . The server uses at most $2n-1$ bits to encode the shape of the tree and at most $n(\lfloor \log N \rfloor + 1)$ bits to encode the assignment of messages to leaves; on average, the client sends about $H + R + \frac{n(\lfloor \log N \rfloor + 3)}{m} + 1$ or $H + R + \frac{n(\lfloor \log N \rfloor + 3)}{m} + 2$ bits, depending on which dynamic Huffman coding algorithm is used [16,15], where R is the redundancy achieved by Huffman coding on S.

Adler and Maggs showed any protocol that uses a single round per message must have a large bound on either the number of bits the server sends, or the number the clients send. TreeQuery is an extreme case — the number of bits the server sends is very large and the number of bits the clients send is nearly minimum — but we can obtain tradeoffs with another protocol, ListQuery.

For *ListQuery*, the server keeps a list of the possible messages, in non-increasing order by their frequency in the prefix of S it has received so far.

It uses an data structure due to Knuth [12] for maintaining a list of integers in non-increasing order, with $O(1)$-time increments. For each message s_i, the server sends the client the first $\lfloor N^{1/k} \rfloor$ messages in the list, where $k > 1$ is a parameter. If s_i is the rth in this sublist, the client responds with 1 followed by the codeword for r in the delta code [4]; if s_i is not in this sublist, it responds with 0 followed by the message.

Theorem 3. *Suppose some clients send S to a server using ListQuery with parameter $k > 1$. For each message, the server sends at most $\lfloor N^{1/k} \rfloor (\lfloor \log N \rfloor + 1)$ bits and, on average, the client sends $kH + \frac{n \log N}{m} + O(1)$ bits.*

Proof. Suppose the client's message s_i appears rth in the sublist it receives from the server; then $\#_{s_i}(s_1, \ldots, s_{i-1}) \leq (i-1)/r$. The codeword for r in the delta code has length $\lfloor \log r \rfloor + 2\lfloor \log(\log r + 1) \rfloor + 1$ which, since $k > 1$, is bounded by $k \log r + O(1)$. Thus, the client sends

$$k \log \frac{i-1}{\#_{s_i}(s_1, \ldots, s_{i-1})} + O(1)$$

bits. Now suppose s_i does not appear in the sublist; then $\#_{s_i}(s_1, \ldots, s_{i-1}) \leq (i-1)/N^{1/k}$. Thus, the client sends

$$\min \left(k \log \frac{i-1}{\#_{s_i}(s_1, \ldots, s_{i-1})},\ \log N \right) + O(1)$$

bits.

Therefore, by Lemma 1 and straightforward calculation, the average number of bits a client sends is $kH + \frac{n \log N}{m} + O(1)$. □

As an aside, we note each tree or list the server sends can be viewed as encoding an approximation of the messages' distribution so far. From this perspective, TreeQuery and ListQuery are applications of results in compressing probability distributions [7].

4 Move-to-Front-and-Truncate

ListQuery is reminiscent of Bentley, Sleator, Tarjan and Wei's Move-to-Front (MTF) compression algorithm [2]. To encode S, MTF keeps a list of the possible messages, in increasing order by the time since their last occurrence. For each message s_i, if s_i is the rth message in the list, then MTF records the codeword for r in the delta code and moves s_i to the front of the list. Bentley *et al.* proved MTF encodes S using $(H + o(H))m + n \log N$ bits. In fact, if the messages' distribution changes, MTF may use significantly fewer than H bits.

Inspired by MTF, we modify ListQuery to obtain *Move-to-Front-and-Truncate* (MFT). The server keeps a list of the $\lfloor N^{1/k} \rfloor$ most recent distinct messages, where $k > 1$ is a parameter, in increasing order by the time since their last occurrence. For each message s_i, the server sends the client the list. If s_i is the

rth in the list, the client responds with 1 followed by the codeword for r in the delta code [4]; if s_i is not in the list, it responds with 0 followed by the message. The server then moves s_i to the front of the list and, if that lengthens the list, deletes the last element.

Theorem 4. *Suppose some clients send S to a server using MFT with parameter $k > 1$. For each message, the server sends $\lfloor N^{1/k} \rfloor (\lfloor \log N \rfloor + 1)$ bits and, on average, the client sends $kH + \frac{n \log N}{m} + O(1)$ bits.*

Proof. Suppose the client's message s_i is the first occurrence of that distinct message a in S; then it responds with $\log N + O(1)$ bits. Now suppose s_i is not the first occurrence of a and let s_h be the preceding occurrence; the client responds with $\log(i-h) + 2 \log \log(i-h) + O(1)$ bits; since $k > 1$, this number is bounded by $k \log(i-h) + O(1)$.

Let $s_{i_1}, \ldots, s_{i_{\#_a(S)}}$ be the occurrences of a in S. The clients send a total of

$$\begin{aligned} &\log N + \sum_{j=2}^{\#_a(S)} k \log(i_j - i_{j-1}) + O(\#_a(S)) \\ &\leq \log N + \#_a(S) k \log \frac{m}{\#_a(S)} + O(\#_a(S)) \end{aligned}$$

bits to communicate these messages. Summing over the distinct elements in S, the clients send $kHm + n \log N + O(m)$ bits in total; the average number of bits a client sends is $kH + \frac{n \log N}{m} + O(1)$. □

5 Conclusions

We have shown Watkinson, Adler and Fich's Bit-Efficient Split protocol [18] can easily be extended to situations when the messages' distribution changes over time. We have also given three dynamic asymmetric communication protocols which require only one round of communication for each message. We feel the last of the three, MFT, is likely the most practical for situations when the number of possible messages is not too large and the number of rounds must be as small as possible.

We leave as future work investigating if other asymmetric communication protocols can be made dynamic. We are particularly interested whether there is a way to keep the number of bits the server sends small, while neither increasing the number of rounds for each message to more than a constant nor increasing any party's computation to more than proportional to the number of bits it sends and receives nor making assumptions about how the messages are generated.

References

1. M. Adler and B. M. Maggs. Protocols for asymmetric communication channels. *Journal of Computer and System Sciences*, 64(4):573–596, 2001.
2. J.L. Bentley, D.D. Sleator, R.E. Tarjan, and V.K. Wei. A locally adaptive data compression scheme. *Communications of the ACM*, 29(4):320–330, 1986.
3. P. Bose, D. Krizanc, S. Langerman, and P. Morin. Asymmetric communication protocols via hotlink assignments. *Theory of Computing Systems*, 36(6):655–661, 2003.
4. P. Elias. Universal codeword sets and representations of the integers. *IEEE Transactions on Information Theory*, 21(2):194–203, 1975.
5. N. Feamster, M. Balazinska, G. Harfst, H. Balakrishnan, and D. Karger. Infranet: Circumventing web censorship and surveillance. In *Proceedings of the 11th USENIX Security Symposium*, pages 247–262, 2002.
6. N. Feamster, M. Balazinska, W. Wang, H. Balakrishnan, and D. Karger. Thwarting web censorship with untrusted messenger discovery. In *Proceedings of the 3rd International Workshop on Privacy Enhancing Technologies*, pages 125–140, 2003.
7. T. Gagie. Compressing probability distributions. Submitted.
8. T. Gagie. Sorting a low-entropy sequence. Submitted to this conference.
9. T. Gagie. Dynamic Shannon coding. In *Proceedings of the 12th European Symposium on Algorithms*, pages 359–370, 2004.
10. S. Ghazizadeh, M. Ghodsi, and A. Saberi. A new protocol for asymmetric communication channels: Reaching the lower bounds. *Scientia Iranica*, 8(4):297–302, 2001.
11. D. A. Huffman. A method for the construction of minimum redundancy codes. *Proceedings of the IERE*, 40(9):1098–1101, 1952.
12. D.E. Knuth. Dynamic Huffman coding. *Journal of Algorithms*, 6(2):163–180, 1985.
13. E.S. Laber and L.G. Holanda. Improved bounds for asymmetric communication protocols. *Information Processing Letters*, 83(4):205–209, 2002.
14. K. Mehlhorn. A best possible bound for the weighted path length of binary search trees. *SIAM Journal on Computing*, 6(2):235–239, 1977.
15. R.L. Milidiú, E.S. Laber, and A.A. Pessoa. Bounding the compression loss of the FGK algorithm. *Journal of Algorithms*, 32(2):195–211, 1999.
16. J.S. Vitter. Design and analysis of dynamic Huffman codes. *Journal of the ACM*, 34(4):825–845, 1987.
17. W. Wang. Implementation and analysis of the Infranet anti-censorship system. Master's thesis, Massachusetts Institute of Technology, 2003.
18. J. Watkinson, M. Adler, and F. Fich. New protocols for asymmetric communication channels. In *Proceedings of the 8th International Colloquium on Structural Information and Communication Complexity*, pages 337–350, 2001.

A Proof of Lemma 1

Lemma 1.

$$\sum_{i=1}^{m} \log \left(\frac{i-1}{\max\left(\#_{s_i}(s_1,\ldots,s_{i-1}),1\right)} \right) \leq (H + O(1))m \ .$$

Proof. Let

$$\begin{aligned} C &= \sum_{i=1}^{m} \log \left(\frac{i-1}{\max\left(\#_{s_i}(s_1,\ldots,s_{i-1}),1\right)} \right) \\ &< \log(m!) - \sum_{i=1}^{m} \log \max\left(\#_{s_i}(s_1,\ldots,s_{i-1}),1\right) \ . \end{aligned}$$

If s_i is the jth occurrence of a in S, then

$$\log \max\left(\#_{s_i}(s_1,\ldots,s_{i-1}),1\right) = \begin{cases} 0 & \text{if } j = 1, \\ \log(j-1) & \text{if } j > 1; \end{cases}$$

thus,

$$\begin{aligned} C &< \log(m!) - \sum_{a \in S} \sum_{j=2}^{\#_a(S)} \log(j-1) \\ &= \log(m!) - \sum_{a \in S} \log(\#_a(S)!) + \sum_{a \in S} \log \#_a(S) \\ &\leq \log(m!) - \sum_{a \in S} \log(\#_a(S)!) + n \log \frac{m}{n} \\ &= \log(m!) - \sum_{a \in S} \log(\#_a(S)!) + O(m) \ . \end{aligned}$$

By Stirling's Formula,

$$x \log x - x \ln 2 < \log(x!) \leq x \log x - x \ln 2 + O(\log x) \ ,$$

so

$$C \leq m \log m - m \ln 2 - \sum_{a \in S} \left(\#_a(S) \log \#_a(S) - \#_a(S) \ln 2 \right) + O(m) \ .$$

Since $\sum_{a \in S} \#_a(S) = m$,

$$C \leq \sum_{a \in S} \#_a(S) \log \frac{m}{\#_a(S)} + O(m) = (H + O(1))m \ .$$

□