Dynamic Asymmetric Communication

Travis Gagie

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

Abstract. We present four new asymmetric communication protocols, with which a server with high bandwidth can help clients with low bandwidth send it messages. Three of our protocols are the first to use only a single round of communication for each message. Unlike previous authors, we do not assume the server knows the messages' distribution.

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 [3] 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 this assumption is often both unwarranted and, fortunately, unnecessary.

Suppose a number of clients want to send messages to a server. At any point, the server knows all the messages it has received so far; each client only knows its own messages and does not overhear communication between other clients and the server. Thus, the server may be able to construct a good code but the clients individually cannot. Adler and Maggs assumed the server, after receiving a sample of messages, can accurately estimate the distribution of *all* the messages. This assumption let them simplify the problem: Can the server help a *single* client send it a message drawn from a distribution known to the server? Given a representative sample of messages and a protocol for this simpler problem, the server can just repeat the protocol for each remaining message. In fact, it can even do this in parallel.

Adler and Maggs gave protocols for the simpler problem in which the server uses its knowledge to reduce the expected number of bits the client sends to roughly the entropy of the distribution. Their work has been improved and extended by several authors [17, 10, 13, 5], whose results are summarized in Table 1,[1] and used in the Infranet anti-censorship system [7, 8]. However, while

[1] Table 1 does not include a recent paper by Adler [1], in which he considered a harder version of the original problem with many clients: Can the server take advantage of correlations between messages? He showed it can, but used the even stronger assumption that the server knows the probability distribution over entire sequences of messages.

P. Flocchini and L. Gąsieniec (Eds.): SIROCCO 2006, LNCS 4056, pp. 310–318, 2006.

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 [3], Watkinson, Adler and Fich [17], Ghazizadeh, Ghodsi and Saberi [10] and Bose, Krizanc, Langerman and Morin [5] 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.* but, to be consistent with the data compression literature (particularly [4]), we use a different notation.

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

implementing Infranet, Wang [16] found the distribution of the messages (webpage requests) changed over time — the sample was unreliable.

We return to the original problem but without the assumption of a representative sample. We present and analyze four asymmetric communication protocols based on techniques from data compression for handling changing and unknown distributions. Our results are summarized in Table 2. To make it easier to compare our results and previous results, we show the average cost of each message; notice H is same in both tables, because the distribution of messages in a representative sample is the same as in the whole sequence. In Section 2 we give a dynamic version of Watkinson, Adler and Fich's Bit-Efficient Split protocol [17]. In Section 3, we present and analyze our TreeQuery and ListQuery protocols, which are the first to use only a single round of communication for each message. This is desirable because, as Adler, Demaine, Harvey and Pătraşcu [2] wrote:

> Any time savings obtained from reducing the number of bits sent by the client could easily be lost by the extra latency cost induced by multiple rounds in the protocol, particularly in long-distance networks, such as satellites, where communication has very high latency.

Finally, in Section 4 we show how Bentley, Sleator, Tarjan and Wei's Move-to-Front compression algorithm [4] can be turned into an elegant single-round asymmetric communication protocol, QueueQuery. Our protocols can be implemented so that each party's computation is proportional to the number of bits it sends and receives.

Consider an everyday example of asymmetric communication — placing a call on a cellular telephone. Because reading information from the phone's display is much faster than typing that information on its keypad, we can say the phone has greater bandwidth than the user. One of the ways the phone helps the user place calls faster is by storing a list of recently called numbers. This feature is common and frequently used, even though the phone may only store the 10 most recently called numbers, which demonstrates the advantage of dynamic

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 $k > 1$.

Protocol	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	$\lfloor N^{1/k} \rfloor \lceil \log N \rceil$	$kH + \frac{n \log N}{m} + O(1)$	1
QueueQuery	$\lfloor N^{1/k} \rfloor \lceil \log N \rceil$	$kH + \frac{n \log N}{m} + O(1)$	1

asymmetric communication protocols; it would be difficult or impossible to accurately estimate the called numbers' distribution from a preliminary sample because, for most users, that distribution changes over time. Let m be the number of calls made from the phone, n be the number of distinct phone numbers called, H be the entropy of the called numbers' distribution, and N be the number of phone numbers in the world. If the phone only stores the 10 most recently called numbers, then it displays about $10 \log_{10} N$ digits per call. To call the ith number in this list, the user types about $\log_{10} i$ digits and, to call a number not stored, he or she types about $\log_{10} N$ digits. Notice this takes a single round. We speculate this protocol is, in practice, so efficient that many users type far fewer than $H / \log_{10} 2 + \frac{n \log_{10} N}{m}$ digits on average per call. Unfortunately, in theory, it is almost useless — if the user calls 11 numbers in turn over and over, then $H / \log_{10} 2 = \log_{10} 11 \approx 1.04$ but he or she has to type every number in full.

The drawback of our single-round protocols is the large bound on the number of bits the server sends; this seems unavoidable if we want to prove good upper bounds. For the simpler problem with a single client and message and the distribution known to the server, Adler and Maggs showed that single-round protocols in which the client sends $O(H+1)$ bits cannot have an $N^{o(1)}$ upper bound on the number of bits the server sends. Adler, Demaine, Harvey and Pătraşcu showed that protocols that use $o\left(\frac{\log \log N}{\log \log \log N}\right)$ rounds with high probability and in which the client sends $O(H + 1)$ bits cannot have a $2^{(\log N)^{1-\epsilon}}$ upper bound on the number of bits the server sends, for any $\epsilon > 0$.[2]

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

[2] This does not contradict the second row of Table 1; Adler and Maggs' second protocol uses $O(1)$ rounds in the expected case but not with high probability.

in S, $\#_a(S)$ is a's frequency in S and log means $\log_2$; i.e., H is the entropy of the messages' distribution (sometimes called the 0th-order empirical entropy of S).

We now present Watkinson, Adler and Fich's *Bit-Efficient Split* protocol [17]. For the moment, assume the server has a representative sample and uses it 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. Gilbert and Moore's algorithm [11], for example, will build such a tree in linear time. For each message s_i, the server starts at the root of T and descends to the leaf v of T storing s_i, as follows. At each proper ancestor u of v, the server sends the active client the message a stored at u; if $a \geq s_i$, then the client responds with 0 and the server descends to u's left child; if $a > s_i$, then the client responds with 1 and the server descends to u's right child. By *active* we mean the client currently sending its message to the server. Without loss of generality, assume messages are $\lceil \log N \rceil$ bits long; otherwise, we use their indices. For $a \in S$, the server descends $\#_a(S)$ times to the leaf storing a. Thus, there are at most

$$\sum_{a \in S} \#_a(S) \left(\left\lceil \log \frac{m}{\#_a(S)} \right\rceil + 1 \right) < (H+2)m$$

rounds. During each round, the server sends $\lceil \log N \rceil$ bits and the active client sends 1 bit.

In contrast, our *Dynamic Bit-Efficient Split* (DBES) protocol does not require the server to know the distribution beforehand. Next, we give a simple implementation of DBES; for each message, the server sends $(H + O(1)) \log N$ bits, on average, and performs $O(N)$ computations. It is possible to reduce the number of computations the server makes to $O((H+1) \log N)$ using a technique developed for dynamic alphabetic coding [9].[3]

For each message s_i, the server builds a leaf-oriented binary search tree T_i on all N 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 $\sum_a \left(\#_a(s_1, \ldots, s_{i-1}) + 1/N\right) = i$.).

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, if the server reaches an internal node at depth $\lceil \log i \rceil + 1$, then it knows the active client's message must be one it has not seen before. In the latter case, to cut short the protocol and save rounds, the server signals the client by sending the same message twice (notice it never does this otherwise); the client responds with s_i. Our analysis relies on the following technical lemma.

[3] Gilbert and Moore's algorithm builds T as a binary trie, with the path to the leaf storing a labeled by a prefix of the binary representation of $\sum_{a'<a} \frac{\#_{a'}(S)}{m} + \frac{\#_a(S)}{2m}$; we can use an augmented splay-tree [15] as a dynamic partial-sum data structure to implicitly represent T, and update it as frequencies change.

Lemma 1. $\displaystyle\sum_{i=1}^{m} \log \frac{i}{\max\left(\#_{s_i}(s_1, \ldots, s_{i-1}), 1\right)} < (H+2)m.$

Proof. Shannon [14] showed that, once we have recorded the frequency of each distinct message in S, we can encode S in less than $(H+1)m$ bits. However, the frequencies tell us nothing about how the messages are ordered in S; since there are $m!/\prod_{a\in S} \#_a(S)!$ possible orderings,

$$(H+1)m > \log \frac{m!}{\prod_{a\in S} \#_a(S)!} = \sum_{i=1}^{m} \log i - \sum_{a\in S} \log(\#_a(S)!) \ .$$

Notice

$$\begin{aligned}
&\sum_{a\in S} \log(\#_a(S)!) \\
&= \sum_{a\in S} \sum_{j=1}^{\#_a(S)-1} \log j + \sum_{a\in S} \log \#_a(S) \\
&= \sum_{a\in S} \sum_{s_i=a} \log \max\left(\#_{s_i}(s_1, \ldots, s_{i-1}), 1\right) + \sum_{a\in S} \log \#_a(S) \\
&= \sum_{i=1}^{m} \log \max\left(\#_{s_i}(s_1, \ldots, s_{i-1}), 1\right) + \sum_{a\in S} \log \#_a(S) \ ,
\end{aligned}$$

so

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

Since $\sum_{a\in S} \log \#_a(S) \le n \log \frac{m}{n} < m$, the claim follows. □

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 active 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

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

times so, by Lemma 1, there are a total of $(H+O(1))m$ rounds for all m messages. The server sends $\lceil \log N \rceil$ 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 round except the last, when it may send $\lceil \log N \rceil$ 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 the oldest 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 the active 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 Gilbert and Moore'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 active 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. □

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. If the server stores the list in a standard balanced binary-search tree implementation of a priority-queue, then updating it takes $O(\log N)$ time after each message.

For each message s_i, the server sends the active client the first $\lfloor N^{1/k} \rfloor$ messages in the list, where $k > 1$ is a parameter. The client makes a single pass through this sublist and, if s_i is rth in the sublist, responds with 1 followed by the codeword for r in Elias' delta code [6]; if s_i is not in the sublist, the client responds with 0 followed by s_i. The codeword for $r \geq 1$ in the delta code consists of a self-delimiting, $(2\lfloor \log(\log r + 1) \rfloor + 1)$-bit encoding of $\lfloor \log r \rfloor + 1$ followed by the $(\lfloor \log r \rfloor + 1)$-bit binary representation of r with the leading 1 removed.

Theorem 3. *Suppose some clients send S to a server using ListQuery with parameter $k > 1$. For each message, the server sends at most $\lfloor N^{1/k} \rfloor \lceil \log N \rceil$ bits and, on average, the active 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 $r \leq N^{1/k}$ and $\#_{s_i}(s_1, \ldots, s_{i-1}) \leq (i-1)/r$. Since $k > 1$, the length of the codeword for r in the delta code — i.e., $\lfloor \log r \rfloor + 2\lfloor \log(\log r + 1) \rfloor + 1$ — is bounded by

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

Now suppose s_i does not appear in the sublist; then $\#_{s_i}(s_1, \ldots, s_{i-1}) \leq (i-1)/N^{1/k}$, so $\log N \leq k \log \frac{i-1}{\#_{s_i}(s_1,\ldots,s_{i-1})}$. Again, the number of bits the client sends is bounded by

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

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

4 QueueQuery

ListQuery is reminiscent of Bentley, Sleator, Tarjan and Wei's Move-to-Front (MTF) compression algorithm [4]. To encode S, MTF keeps a list of the possible messages in increasing order by the time since their last occurrence; e.g., the most recent message is first in the list. 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. Notice MTF's list is essentially a reversed queue; the tail is first, the head is last and when messages occur, they move to the tail.

Inspired by MTF and our cell phone example in the introduction, we modify ListQuery to obtain *QueueQuery*. The server keeps a queue of the $\lfloor N^{1/k} \rfloor$ most recent distinct messages, where $k > 1$ is a parameter. This is the only thing the server stores, so QueueQuery is more space-efficient than our previous protocols.

For each message s_i, the server sends the active client this queue. The client makes a single pass through the queue and, if s_i is rth from the tail, responds with 1 followed by the codeword for r in the delta code; if s_i is not in the queue, it responds with 0 followed by s_i. The server then puts s_i at the tail and, if that lengthens the queue, dequeues the message at the head.

Notice that, if s_i is in the queue, then apart from the leading 1 indicating s_i's presence the client sends as many bits as MTF would use to encode s_i. If s_i is not in the queue, then apart from the leading 0 indicating s_i's absence the client sends at most about k times as many bits as MTF would use; i.e., $\lceil \log N \rceil$ instead of at least $\lfloor \log N^{1/k} \rfloor + 2\lfloor \log(\log N^{1/k} + 1) \rfloor + 1$ bits. Thus, although our analysis below of QueueQuery is self-contained, it is naturally very similar to that of MTF.

Theorem 4. *Suppose some clients send S to a server using QueueQuery with parameter $k > 1$. For each message, the server sends $\lfloor N^{1/k} \rfloor \lceil \log N \rceil$ bits and, on average, the active 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. If the number

of distinct messages in $s_h, \ldots, s_{i-1}$ is greater than $N^{1/k}$, then s_i is no longer in the queue and the client responds with $\log N + O(1)$ bits; since $i - h$ must be greater than $N^{1/k}$, this is bounded by $k \log(i-h) + O(1)$. Otherwise, s_i is still in the queue and the client responds with $\log(i-h) + 2\log\log(i-h) + O(1)$ bits which, since $k > 1$, is also 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 with these messages send a total of

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

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

Acknowledgments

Many thanks to Faith Ellen Fich, Giovanni Manzini and Charlie Rackoff, who supervised this research, and to the anonymous referees, for helpful comments.

References

1. M. Adler. Collecting correlated information from a sensor network. In *Proceedings of the 16th Symposium on Discrete Algorithms*, pages 479–488, 2005.
2. M. Adler, E.D. Demaine, N.J.A. Harvey, and M. Pătraşcu. Lower bounds for asymmetric communication complexity and distributed source coding. In *Proceedings of the 17th Symposium on Discrete Algorithms*, pages 251–260, 2006.
3. M. Adler and B.M. Maggs. Protocols for asymmetric communication channels. *Journal of Computer and System Sciences*, 64(4):573–596, 2001.
4. 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.
5. P. Bose, D. Krizanc, S. Langerman, and P. Morin. Asymmetric communication protocols via hotlink assignments. *Theory of Computing Systems*, 36(6):655–661, 2003.
6. P. Elias. Universal codeword sets and representations of the integers. *IEEE Transactions on Information Theory*, 21(2):194–203, 1975.
7. 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.
8. 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.
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. E.N. Gilbert and E. Moore. Variable-length binary encodings. *Bell System Technical Journal*, 38:933–968, 1959.
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. C.E. Shannon. A mathematical theory of communication. *Bell System Technical Journal*, 27:379–423, 623–656, 1948.
15. D.D. Sleator and R.E. Tarjan. Self-adjusting binary search trees. *Journal of the ACM*, 32:652–686, 1985.
16. W. Wang. Implementation and security analysis of the Infranet anti-censorship system. Master's thesis, Massachusetts Institute of Technology, 2003.
17. 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.