On Disseminating Information Reliably Without Broadcasting

by

Noga Alon
Amnon Barak
Udi Manber

Computer Sciences Technical Report #621

December 1985

ON DISSEMINATING INFORMATION RELIABLY WITHOUT BROADCASTING

Noga Alon[†]
Department of Mathematics
Tel Aviv University / MIT

Amnon Barak[‡]
Department of Computer Science
Hebrew University

Udi Manber[§]
Department of Computer Science
University of Wisconsin - Madison
Madison, WI 53706

November 1985

ABSTRACT

A general scheme for collecting information in a multicomputer system is presented. The scheme allows data to be exchanged efficiently and quickly among many machines without broadcasting. In addition, certain operations can be performed on the data while it is being collected. Several applications to distributed computing are discussed. As a byproduct of the combinatorial part of the paper, a problem of Babai and Erdős in combinatorial group theory is solved.

† Supported in part by the United States Air Force Office of Scientific Research under grant AFOSR-82-0326.

‡ Supported in part by the United States Air Force Office of Scientific Research under grant AFOSR-85-0284.

§ Supported in part by the National Science Foundation under grants MCS-8303134 and DCR-8451397.

1. INTRODUCTION

The problem of exchanging data in multicomputer systems is an important basic problem. There are many cases in which the movement of data constitutes the main cost of the computation. In this paper we present a scheme for disseminating information among several machines connected through a local area network. We consider a model of computation based on existing systems. We first reduce the problem in this model to a combinatorial problem. We then present a simple elegant solution, using latin squares and finite fields, for the case when the number of machines is a power of 2. The solution is optimal and easy to implement. For the other cases we present schemes based on random generation of permutations and show that these schemes are optimal to within a constant. The combinatorial problems we encounter are interesting on their own. In particular, we solve a problem raised by Babai and Erdös [BE82] about the representation of all the elements of a group as short products (see theorem 6.8).

Consider a multicomputer system having N independent computers and workstations connected through a local area communication network. There are two major types of local area networks in operation today: Token Ring networks, and Ethernets (see for example [Ta81]). Both of these networks are based on a broadcast medium so that each message reaches the whole network. All machines look at the address of the message, and only the destination reads the message. As a result, the cost of sending a message is independent of the participants.

Consider now a problem of computing a simple function of variables that reside at the different machines (e.g., computing the average), in a way that the outcome becomes known to all machines. A straightforward solution is to send all the data to one machine, perform the computation there, and broadcast the results. This approach produces a bottleneck and it is obviously not very robust. Another approach is to simply broadcast all the data to all machines and have all of them perform the computation. The main problem with this approach is that the cost associated with sending messages is not only the transmission delay but also the processing time for each machine. If we use broadcast too often we may lose in processing time what we gain in delay. Another major problem of using broadcast even in broadcast media is acknowledgments. This is especially critical in the Ethernet since, unless we are careful, after one broadcast all machines will try to acknowledge at the same time causing significant additional delay due to transmission conflicts.

In this paper we present a scheme to "broadcast without broadcasting". We use repeated forwarding and packaging of messages to disseminate information. We show that broadcasting can be achieved quickly without physical broadcast and with some measure of reliability. (For the rest of this paper we use the word broadcast to denote our method of disseminating information rather than an actual broadcast.) We also show that one can use this scheme of forwarding messages to design efficient algorithms for certain problems. We view this scheme as part of a high level algorithm geared towards specific applications such as load balancing. We do not suggest to implement it as part of an all-purpose communication network. We assume a local area network in which the cost of messages between any two machines is the same. We do not assume any other special capabilities. For a different, more efficient, scheme that requires additional hardware see Livny and Manber [LM85]. The emphasis in this paper is on the algorithmic aspects of the scheme.

2. MOTIVATION AND DEFINITIONS

The original motivation for this paper was a problem in load balancing [BD84]. The main part of the problem was to find the average load of N independent machines with little overhead for the communication. The following communication pattern was used in the solution. Communication was divided into *rounds*, in which each machine sends exactly one message to another machine. The cost of processing messages is thus divided equally among the machines. (Communication in rounds is especially natural if the access to the network is allocated in a round robin fashion, independently of the activity of the machines. This type of multiaccess technique is called a *fixed assignment technique* [To80].) The destination of each message was selected at random. Each message incorporates

the current knowledge of that machine. In [DB84] it was proved that this scheme allows broadcasting in $(1+\ln 2)\log_2 N \approx 1.693\log_2 N$ expected number of rounds. Since all machines initiate broadcasts (with their load), after $1.693\log_2 N$ expected number of rounds all of them can incorporate the loads of all other machines in their computation (a description of the computation is given in section 5). This approach was used in the MOS distributed operating system, which supports load balancing by dynamic process migration [BL85]. In this paper we improve on these results in several ways. In particular, we describe, for all $N=2^n$, a deterministic algorithm for broadcasting in $\log_2 N$ rounds, which is the smallest possible number of rounds.

There are several drawbacks in the probabilistic approach used in [BD84]. First, the probabilistic nature of the algorithm implies that the exact number of rounds required for a completion of a "broadcast" is unknown. Moreover, even though the expected number of rounds is small, it is still more than the lower bound (of $\log_2 N$). Second, the probabilistic algorithm is not robust. Since messages are received at random there is no way to detect whether all machines cooperate. If several machines fail during (or before the start of) the broadcasting and if the broadcasting is used as part of an algorithm (such as computing the average load) then the outcome may be imprecise. In a deterministic algorithm each machine expects messages from specific machines, and if a message does not arrive it can take preventive actions (assuming that messages are reasonably [not necessarily completely] synchronized). It is also important to make the algorithm as symmetric as possible to prevent greater dependency on a small number of machines. Third, as was mentioned in the introduction, processing time of each message may be significant; hence, it is important that each machine receives only one message in each round and not only transmits only once in each round.

The three issues above led us to develop the following scheme for a solution to the problem. The solution is completely deterministic, optimal for $N=2^n$, robust, and symmetric. We start with a combinatorial definition of the problem.

Denote the machines by $M_0, M_1, \ldots, M_{N-1}$. In each round k each machine sends a message to another machine. Round k can thus be described by a vector $R_k = (a_0, a_2, \ldots, a_{N-1})$, where $a_i = j$ iff machine M_i sends a message to machine M_j in round k. In order to distribute the work evenly among machines in a round we further assume that each machine receives one message in each round. Hence, each round vector corresponds to a permutation. Our second and most important objective is to achieve fast dissemination of information. Define the *broadcast set* of machine i for rounds k to $k+t$ as follows. At the beginning of round k only i belongs to the broadcast set. Assuming that at the beginning of round $k+j$ $(j \geq 0)$ the broadcast set is B, then after round $k+j$ all machines that receive messages from any machine in B are added to the broadcast set. The broadcast set for rounds k to $k+t$ is the broadcast set after round $k+t$. (We assume here that machines forward all the information they have. In many cases this will not be required; we discuss this point in detail in section 5.) Optimally, the broadcast set will be doubled in each round. The *broadcast time* for machine i beginning at round k is defined as the minimal t such that the broadcast set for rounds k to $k+t-1$ includes all machines. It is obvious that for N machines the broadcast time is always at least $\lceil \log_2 N \rceil$. We would like to achieve a broadcast time close to $\log_2 N$ for all machines and all starting rounds.

Our third objective is to allow easy determination of machines that failed (or stopped cooperating). This is achieved by requiring that in any $N-1$ consecutive rounds each machine receives messages directly from all other machines. Hence, if the status of machines do not change in $N-1$ rounds then all machines know exactly who failed. It is easy to see that the requirement above implies that every machine transmits to all other machines in any $N-1$ rounds, and hence the order in which every machine transmits is a cyclic permutation of the $N-1$ other machines. This requirement also makes the algorithm more symmetric. If we now build an $N-1 \times N$ matrix such that rows correspond to consecutive round vectors, then each column j is a permutation of the numbers from 0 to $N-1$ except for j. In other words, if we put the identity vector $(0,1,2,\ldots,N-1)$ at the end of the matrix we get a *latin square*. The problem now becomes a combinatorial problem of designing latin squares that minimize the broadcast time as defined above. We call the $N-1 \times N$ latin squares

without the identity vector *truncated latin squares.*

We will show that for any $N = 2^n$ there exists a truncated latin square that always achieves the optimal broadcast time of n. Algorithms that are based on this framework will be discussed. We will also present efficient truncated latin squares for N which is not a power of 2.

3. A REDUCTION OF THE PROBLEM TO A COMBINATORIAL PROBLEM

The broadcast time is defined in terms of an iterative procedure (machines forwarding messages in rounds). This makes it harder to analyze. We reduce the problem to an easier to describe combinatorial problem by restricting the types of truncated latin squares we consider. Let $G = \{g_0, g_1, \ldots, g_{N-1}\}$ be a group of order N with an identity element g_0. It is convenient to think of the integers modulo N as the group, but any group will do. In fact, the group need not be abelian, although, for convenience we use abelian groups terminology. We will use the fact the all the results hold for non-abelian groups to solve the problem of Babai and Erdös in section 6. Let $A = G - \{g_0\}$. For notational convenience we associate each number i, $0 \le i < N$ with the group element g_i, and consider truncated latin squares whose entries are the g_j's, and whose columns are indexed by G. The column indexed by g_0 of the latin square is, by definition, a permutation of A. Denote this permutation by $\pi = (\pi_1, \pi_2, \ldots, \pi_{N-1})$. For $0 < i < N$, we set the g_i'th column to be a shift by g_i of the first column. Namely, the (j, g_i)'th entry equals $g_i + \pi_j$. The group properties guarantee that this is indeed a truncated latin square. It is easier to compute the broadcast time for such latin squares than for general ones. This is done in the next two simple theorems.

THEOREM 3.1. Let $G, A, \pi = (\pi_1, \pi_2, \ldots, \pi_{N-1})$ be as above. Then the broadcast set for machine g_k for rounds j to $j+t$ is $\{s \in G \mid s = g_k + \sum_{i=0}^{t} c_i \pi_{j+i}, c_i \in \{0,1\}\}$, where the subscripts of π are taken cyclically in the range $(1,,N-1)$.

PROOF: We apply induction on t. At round j machine g_k sends a message to machine $g_k + \pi_j$, and hence the broadcast set for $t=0$ is $\{g_k, g_k + \pi_j\}$, proving the theorem for $t=0$. Assuming the result for $t-1$ we prove it for t, $(t>0)$. Let M be the broadcast set for g_k for rounds j to $j+t-1$. If $s \in M$, then at round $j+t$, machine s sends a message to machine $s + \pi_{j+t}$. Thus the new broadcast set is $M \bigcup (M + \pi_{j+t})$, and the theorem follows. □

COROLLARY 3.2. The broadcast time is independent of the machine that initiated the broadcast. It depends only on the starting round number j, and it is equal to the minimum t such that each $g \in G$ has the form

$$(3.1) \quad \sum_{i=0}^{t-1} c_i \pi_{j+i},$$

where $c_i \in \{0,1\}$ and all subscripts are taken cyclically in the range $(1,,N-1)$.

PROOF: The corollary follows immediately from Theorem 3.1 and the fact that G is a group. □

From now on we will assume that the broadcast is always initiated by machine g_0. Theorem 3.1 gives a clear characterization of broadcast time. We need to find a cyclic permutation π of $G - \{g_0\}$, such that every small number of consecutive elements of the permutation "generate" G in the sense expressed in (3.1). We say that a cyclic permutation has broadcast time t if the corresponding truncated latin square has broadcast time t, i.e., for every j, every $g \in G$ has the form (3.1).

4. OPTIMAL TRUNCATED LATIN SQUARES

As mentioned in Section 2, the broadcast time for N machines is always at least $\lceil \log_2 N \rceil$. By the results of section 3, to obtain such time it is enough to find a group G of order N and a cyclic permutation π of $A = G - \{0\}$, such that any $k = \lceil \log_2 N \rceil$ consecutive elements of π generate G. The following two theorems supply two families of such permutations, using some elementary properties of finite fields (cf. [Ha67] or [He64]).

THEOREM 4.1. Suppose $N = 2^k$, and let G be the additive group of the finite field $GF(2^k)$. Let $g \in G$ be a primitive root of $GF(2^k)$. Then $\pi = (g^0, g^1, \ldots, g^{2^k-2})$ is a cyclic permutation of $GF(2^k) - \{0\}$ whose broadcast time is $k = \log_2 N$.

PROOF: g is the root of a primitive polynomial $p(x)$ of degree k over $GF(2)$. The field $GF(2^k)$ is represented by the residue classes modulo $p(x)$ of the polynomials over $GF(2)$, and g is a generator of the multiplicative group of $GF(2^k)$ (see [Ha67] or [He64]). Therefore, $g^0, g^1, \ldots, g^{N-2}$ is a permutation of $A = GF(2^k) - \{0\}$ and every element $g \in G$ has a unique representation as a polynomial of degree at most $k-1$ over $GF(2)$. To prove the theorem we must show that for every $0 \le j \le 2^k - 2 = N - 2$, every $g \in GF(2^k)$ can be expressed in the form

$$(4.1) \quad g = \sum_{i=0}^{k-1} c_i g^{j+i}, \text{ where } c_i \in \{0,1\}.$$

However, g/g^i is an element of $GF(2^k)$ and thus can be expressed as a polynomial $\sum_{i=0}^{k-1} c_i g^i$ of degree at most $k-1$ over $GF(2)$. This implies (4.1) and completes the proof. □

It is worth noting that once a primitive root g for $GF(2^k)$ is found and the permutation π is constructed, every power g^j of g can be expressed as a vector of length k over $GF(2)$ (representing the corresponding polynomial) and the addition in the group G is just the usual addition of vectors modulo 2. Hence the implementation of the broadcast algorithm is very simple.

THEOREM 4.2. Suppose $N = p$ is a prime and 2 is a generator of the multiplicative group modulo p. Let G be the abelian group Z_p and put $\pi = (2^0, 2^1, \ldots, 2^{p-2})$. Then π is a cyclic permutation of $Z_p - \{0\}$ whose broadcast time is $k = \lceil \log_2 N \rceil$.

PROOF: As before, we must show that for every $0 \le j \le p-2$, every $g \in Z_p$ can be expressed in the form $g = \sum_{i=0}^{k-1} c_i 2^{j+i}$, where $c_i \in \{0,1\}$. However, the element $g/2^j \in Z_p$ has a binary representation $\sum_{i=0}^{k-1} c_i 2^i$, which gives the desired result. □

Although the results above hold only for powers of 2 and certain primes they can be used to design an approximate scheme for any N in the following way. Let N be such that $2^{k-1} < N < 2^k$. Denote by M_1 the set of N machines and by M_2 the set of 2^k machines. Let T denote the first $2^k - N$ machines in M_1. These machines will "take over" the forwarding for the last $2^k - N$ machines in M_2. We "simulate" each round of the solution for M_2 by two semi-rounds with M_1. In a semi-round not all machines may transmit or receive but no machine sends or receives more than one message. The communication pattern will be the same as the one for M_2 except that messages that involve the last $2^k - N$ machines in M_2 will be handled by T. We now show that each round can always be divided into 2 semi-rounds. Each machine in T sends and receives exactly 2 messages and all other machines in M_1 send and receive one message. Assume that all machines in M_1 send and receive 2 messages; this is obviously a worst case (one can always add dummy messages). The problem is to divide the messages into two groups (colors) such that two messages in the same group do not originate from the same machine and do not have the same destination. This can

always be done in linear time (e.g., use Hall-König Theorem [Ha67]).

The approximate scheme described above has many of the advantages of the original scheme. The main drawback is that it takes $2\lceil \log_2 N \rceil$ to broadcast and $2(N-1)$ to receive messages from all other machines.

5. ALGORITHMS BASED ON THE COMMUNICATION PATTERN

Let $N=2^k$ and consider the optimal scheme presented in the previous section. Assume that each machine has a *value* and that all machines start broadcasting their values at the same time. Furthermore, assume that all machines forward all the values they receive. It is clear that after k rounds every machine will know the values of all other machines and thus can perform any algorithm on the values. Overall, there are $N \log_2 N$ messages whose sizes double at every round, and, since the scheme is optimal, no value is ever sent twice to the same machine. This compares favorably with the N^2 messages required to exchange the N values directly. In many cases, especially if the values have short representation, longer messages do not cost significantly more.

There is however a much greater benefit with this scheme since for some algorithms the values need not be accumulated. The algorithms can be performed by the machines as they forward the messages, thus saving time and communication. We present two examples of such algorithms.

First we consider the problem (which motivated this research) of finding the average load in a multicomputer system. Assume that the computation starts every k rounds. In the first round all messages contain one load. In the second round all machines know of one other load in addition to their own. They can average the 2 loads and send that information. After the second round all machines have 2 averages, each of 2 loads, hence they can get the average of 4 loads. The full algorithm is given below.

Algorithm Average

1. at round 1 each machine sends out its load; all messages are sent according to the truncated latin square given in section 4.
2. at round j, $1<j\leq k$, each machine sends out the average of the message (load) it received at round $j-1$ and the message it sent at round $j-1$.

THEOREM 5.1. In Algorithm Average after k rounds all machines have the exact average load.

PROOF: The proof is by induction on the number of rounds.

Induction hypothesis: at the beginning of round j, $1\leq j\leq k+1$, each outgoing message contains the average load of 2^{j-1} distinct machines.

The hypothesis is clearly true for $j=1$, and it implies the theorem since if at round $k+1$ each machine could send the average of 2^k loads then each machine knows the average. Consider round $j+1$ and machine M. At round j M received the average load of 2^{j-1} machines and sent away an average load of 2^{j-1} machines. We only have to prove that these 2^j loads are all distinct. Assume the contrary. Then, there exists one machine $\bar{M}$ such that its original load was involved in 2 messages sent to M. One message at some round $i<j$ (so that M can use it in the message it sent at round j), and another message at round j. However, by the optimality of the broadcast starting from $\bar{M}$ at round 1, every time $\bar{M}$'s message is forwarded it is sent to new machines. Hence, within k rounds, M cannot receive $\bar{M}$'s message twice.

□

Notice that the proof above is independent of the specific truncated latin square; it depends only on the fact that the broadcast is optimal. Algorithm Average computes the average load every $\log_2 N$ rounds. The total number of messages is $N \log_2 N$. If we need the average load more frequently we can start the computation more often and have several computations proceed in parallel. Each

message will contain several averages in different phases of the computation. Messages will be longer but more computation will be done. In many networks the overhead associated with sending a message is high and as a result sending long messages is much more cost effective than sending several shorter messages. In practice having a message with $\log_2 N$ numbers is very reasonable. Such messages allow us to compute the average load in each round (although we get the average load of $\log_2 N$ rounds ago).

Finding averages is not the only efficient algorithm we can perform using our scheme. It is easy to see that finding a maximum or minimum of N variables can be done in exactly the same way. It is interesting to try to identify the type of problems that can be solved in $\log_2 N$ rounds (or just $O(\log n)$) with this scheme. We give here one more such problem - the *majority problem.* The majority problem is to determine, given N *votes* (integers), whether one vote appears more than $N/2$ times (a majority) and if so to find it. It is important, for example, in distributed databases recovery algorithms [TH79]. The algorithm is an extension of the sequential algorithm given by Fischer and Salzberg [FS82].

Algorithm Majority

The algorithm is divided into two phases, each lasting $\log_2 N$ rounds: 1. selection of a candidate, and 2. verification. In the first phase we select a vote that is guaranteed to be the majority provided a majority exists. In the second phase we verify that this vote is indeed a majority. In the first phase each message contains two integers, C (candidate) and M (multiplicity). Initially each machine sets C to be its vote and $M=1$. Let $C_{in}(M_{in})$ be the values machine p received in round j $(1<j<k+1)$, and $C_{out}(M_{out})$ the values it sent in round j. The values C, and M sent out in round $j+1$ are the following.

1. if $C_{in}=C_{out}$ then $C:=C_{in}$; $M:=M_{in}+M_{out}$;
2. if $C_{in}\neq C_{out}$ then there are two cases:

   2a. if $M_{in}\geq M_{out}$ then $C:=C_{in}$; $M:=M_{in}-M_{out}$;

   2b. if $M_{in}<M_{out}$ then $C:=C_{out}$; $M:=M_{out}-M_{in}$;

The idea behind the algorithm is the following. If a majority exists and if, given two different votes, we eliminate both of them and continue doing so until only one vote remains, then this vote must be the majority since there are not enough other votes to eliminate it. Hence, if, after $\log_2 N$ rounds, the multiplicity of the candidate C is not zero then C is a candidate (and the only possible one). The second phase consists of counting how many times C appears. That can be easily achieved by initially sending out a 1 (if your vote equals C) or 0 (otherwise), and in each of the other rounds just sum the values.

If N is not a power of 2 then the approximate scheme described in the previous section can be used. It takes twice the time but the algorithms can be very similar. The only difference is that some messages are "dummy" messages used only to simulate the optimal scheme. We have to make sure that these messages do not contribute to the algorithm. In the algorithm for finding the average one can assign 0 loads to the non existing machines and scale everything at the end. In the algorithm for finding the majority it is sufficient to assign multiplicity of 0 to non existing machines.

6. RANDOM TRUNCATED LATIN SQUARES

In this section we prove that a random permutation of the non-identity elements of any group G of order N produce, with high probability, a truncated latin square whose broadcast time is $O(\log n)$. Empirical results (for the group Z_N) suggest that for large number of machines the constant is less than 2 (see section 7). Hence, this may be faster than the approximate scheme described in the previous section. While the choice of the truncated latin square is random, once it is made the

algorithm is completely deterministic, thus the broadcast time is fixed and can easily be determined. As a byproduct of our method we settle a problem of Babai and Erdős in combinatorial group theory. We start with some technical lemmas. Although we use here additive notation for convenience, we do not assume that the groups in this section are abelian.

LEMMA 6.1. Let G be a group, and let H be a subset of G. Let g be a random element of G. Then the expected value of $|H\cap(H+g)|$ is $|H|^2/|G|$.

PROOF: For every fixed $b\in H$ the number of $g\in G$ such that $b+g\in H$ is precisely the cardinality of $-b+H$, which is $|H|$. Since $|H\cap(H+g)| = \sum_{b\in H} |H\cap\{b+g\}|$ the desired result follows.

□

LEMMA 6.2. Let G,H be as in Lemma 6.1. Then the number of $g\in G$ such that

$$|H\cap(H+g)| > \frac{3|H|^2}{2|G|}$$

is less than $2/3\,|G|$.

PROOF: Immediate from Lemma 6.1.

□

LEMMA 6.3. Let G,H be as in Lemma 6.1. Then the number of $g\in G$ such that

$$|G-(H\cup(H+g))| > \frac{3|\bar{H}|^2}{2|G|}$$

is less than $2/3\,|G|$, where $\bar{H} = G-H$.

PROOF: Notice that $G-(H\cup(H+g)) = (G-H)\cap(G-(H+g)) = \bar{H}\cap(\bar{H}+g)$. Hence, the lemma follows from Lemma 6.2 by replacing H by $\bar{H}$.

□

LEMMA 6.4. Let G,H be as in Lemma 6.1, and suppose $|H| < \frac{1}{2}|G|$, $K\subseteq G$, $|K|\le |G|/6$. The number of $g\in G-K$ such that $|(H\cup(H+g))| \ge 5/4\,|H|$ is at least $|G|/6$.

PROOF: By Lemma 6.2 for at least $(1/3-1/6)\,|G|$ elements $g\in G-K$

$$|H\cap(H+g)| \le \frac{3|H|^2}{2|G|}.$$

Hence, for those elements g,

$$|H\cup(H+g)| \ge 2|H| - |H\cap(H+g)| \ge 5/4\,|H|.$$

□

LEMMA 6.5. Let G,H be as in Lemma 6.1, and suppose $|H| > \frac{1}{2}|G|$, $K\subseteq G$, $|K|\le |G|/6$. The number of $g\in G-K$ such that $|G-(H\cup(H+g))| < 3/4\,|G-H|$ is at least $|G|/6$.

PROOF: By Lemma 6.3 for at least $(1/3-1/6)\,|G|$ elements $g\in G-K$

$$|G-(H\cup(H+g))| \le \frac{3|\bar{H}|^2}{2|G|} \le \frac{3|G-H|}{4}$$

□

Suppose now that G is a group of order N. We say that a sequence $B = (b_1,b_2,\ldots,b_k)$ of elements of G *represents* G if every $g\in G$ has a representation of the form $\sum_{i=1}^{k} c_i b_i$, where $c_i\in\{0,1\}$. Put $k = 36\log_{5/4}N$ and assume that $k\le N/6$.

LEMMA 6.6. The probability that a random sequence $B = (b_1,b_2,\ldots,b_k)$ of k distinct non-identity elements of G does not represent G is less than $N^{-1.1}$.

PROOF: Choose the elements of B one by one. Put $B_i = (b_1,..,b_i)$ and let H_i be the set of all elements of G represented by B_i. Call the choice of $b_{i+1} \in G - B_i$ a *success* if one of the following happens; either $|H_i| \leq \frac{1}{2}|G|$ and $|H_{i+1}| \geq 5/4\,|H_i|$ or $|H_i| > \frac{1}{2}|G|$ and $|G - H_{i+1}| \leq 3/4\,|H_i|$.

By Lemma 6.4 and Lemma 6.5 the probability that b_{i+1} is a success, given all the previous choices, is always at least 1/6. Hence, the expected number of successes in $k = 36\log_{5/4}N$ choices is at least $6\log_{5/4}N$, and we can lower bound the probability for having at least $2\log_{5/4}N \geq \log_{5/4}N + \log_{4/3}N$ successes by using the standard estimate given by Chernoff's inequality (cf. [ES74]) for a binomial distribution with $p = 1/6$. (Notice that this is justified by the preceding remarks although, in fact, we do not have here a binomial distribution.) Hence, the probability of having at least $2\log_{5/4}N$ successes is at least $1 - N^{-1.1}$. However, that many successes guarantee, as can be easily checked, that $H_k = G$, i.e., that B represents G.

□

As an immediate corollary we obtain the following theorem.

THEOREM 6.7. Let G be a group of order N with identity 0, and suppose $k = 36\log_{5/4}N \leq N/6$. Then there is a permutation $\pi = (\pi_0,\pi_1,...,\pi_{N-2})$ of $A = G - \{0\}$ such that for all $0 \leq i < N-1$, the sequence $\pi_i,\pi_{i+1},...,\pi_{i+k-1}$ where the indices are reduced modulo $N-1$, represents G. Moreover, almost all permutations have that property.

Remarks:

1. By a more careful analysis one can reduce the constant in our estimate $36\log_{5/4}N$ considerably. We only wanted to establish $O(\log N)$. It might be that the correct estimate is $\log_2 N + O(\log \log N)$. At the moment, however, we are unable to prove it.
2. By a trivial modification of the proof of Lemma 6.6 (considering multiplicative non-abelian groups), one can prove the following (slightly easier) result;

THEOREM 6.8. There exists a positive constant c such that for every group G of order N and a random choice of (not necessarily distinct) $x_1,x_2,...,x_k \in G$, the probability that the sequence $B = (x_1,x_2,...,x_k)$ represents G tends to 1 while $N \to \infty$ and $k = \lceil c\log N \rceil$. (B represents G if every $g \in G$ has a representation of the form $\prod_{i=1}^{k} b_i^{\epsilon_i}$, where $\epsilon_i \in \{0,1\}$).

This answers in the affirmative a question raised by Babai and Erdös in [BE82].

7. EMPIRICAL RESULTS

7.1. Random permutations

Since the constant in the $O(\log N)$ in the proof of Theorem 4.1 is larger than 2 we ran experiments (by choosing random permutations in the group Z_N) to find the broadcast time for reasonable values of N. We found that for all $N \leq 3200$ the constant is less than 2. As mentioned, it may be possible that the multiplicative constant is 1. Table 1 gives the results of a simulation using random permutations. For each value of N 10 random permutations were tried (the variance was very small). The average and maximum broadcast times over all starting rounds and all trials are given.

N	$\lceil \log_2 N \rceil$	average	maximum
25	5	6.7	8
50	6	8.0	11
100	7	9.2	12
200	8	10.3	13
400	9	11.5	14
800	10	12.8	15
1600	11	14.0	16
3200	12	15.1	18

Table 1: Broadcast time for random truncated latin squares

7.2. Random failures

The results of the previous sections depended on the cooperation of all machines. It is important to see what happens if several machines fail. In this section we present simulation results on the expected broadcast time for random failures. The simulations indicate that the broadcast time remains low even if significant number of machines fail. In the simulations we did not assume any knowledge of failures. One may be able to improve the performance by adaptively changing the communication patterns once machines learn about failures. Table 2 below gives the average and maximum broadcast times for N machines over $N-1$ rounds with different number of random failures. Broadcast time in this case is defined as the time to reach all **non-failing** machines. The variances in the simulation were always less than 10%. The truncated latin squares are the optimal ones presented in section 4. While this data is not enough to predict the asymptotic behavior of the broadcast time with random failures, we believe it gives strong evidence that the broadcast time in our scheme is still pretty low. With a 10% failure rate the broadcast time (for $N \leq 1024$) is no more than double the optimal time and about 40% above optimal time on the average.

number of machines	percentage of failures	average	maximum
64	5	7.0	9
64	10	7.6	11
64	20	8.7	14
128	5	8.4	13
128	10	9.4	13
128	20	10.9	16
256	5	10.0	14
256	10	11.0	15
256	20	12.7	18
512	5	11.4	15
512	10	12.5	17
512	20	14.6	22
1024	5	12.8	17
1024	10	14.1	19
1024	20	16.4	24

Table 2: Broadcast time with random failures

8. CONCLUSIONS AND FURTHER RESEARCH

We introduced a new scheme for achieving broadcasting without physical broadcasts. The scheme can be used for a variety of algorithms. It is easy to implement and it does not require any special support from the operating system or the communication network.

Several open problems remain.

1. Determining the expected broadcast time in the presence of failures. Our simulations indicate that the algorithms can tolerate failures reasonably well. It seems hard to be able to find analytic solutions. One may also be able to improve the broadcast time by using the information about machine failures.
2. Finding the optimal truncated latin squares for N which is not a power of 2 and not a prime. We expect the broadcast time to be very close to $\log_2 N$.
3. Designing more algorithms that use the communication pattern suggested in the paper.

Acknowledgment: We would like to thank L. Babai, E. Bach, R. Finkel, R. Manber, and M. Solomon for fruitful discussions.

REFERENCES

[BD84]
A. Barak and Z. Drezner, "Distributed Algorithms for the Average Load of a Multicomputer," University of Michigan, Computing Research Laboratory, Technical report CRL-TR-17-84, (March 1984).

[BE82]
L. Babai and P. Erdös, "Representation of group elements as short products" in *Theory and Practice of Combinatorics*, (A. Rosa et al eds.), Annals of Discrete Math. **12** (1982), 27-30.

[BL85]
A. Barak and A. Litman, "MOS: A Multicomputer Distributed Operating System," *Software Practice and Experience*, **15** (August 1985), 725-737.

[DB84]
Z. Drezner and A. Barak, "An Probabilistic Algorithm for Scattering Information in a Multicomputer System," University of Michigan, Computing Research Laboratory, Technical report CRL-TR-15-84, (March 1984).

[ES74]
P. Erdös and J. Spencer, "Probabilistic Methods in Combinatorics," Academic Press (1974), New York and London, pp. 18.

[FS82]
M. J. Fischer and S. L. Salzberg, "Finding Majority Among n Votes," Research Report #252, Yale University, Department of Computer Science, October 1982.

[Ha67]
M. Hall Jr., Combinatorial Theory, Blaisdell Publishing Company, London, 1967.

[He64]
I. N. Herstein, Topics in Algebra, Blaisdell Publishing Company, London, 1964.

[LM85]
M. Livny and U. Manber, "Distributed Computation via Active Messages," to appear in *IEEE Transactions on Computers*, Special issue on Distributed Computing, (December 1985).

[TA81]
A. S. Tanenbaum "Computer Networks," Prentice-Hall, 1981.

[TH79]
R. H. Thomas, "A Majority Consensus Approach to Concurrency Control for Multiple Copy Database," *ACM Transactions on Database Systems*, **4** (June 1979), 180-209.

[To80]
F. A. Tobagi, "Multiaccess Protocols in Packet Communication Systems," *IEEE Transactions on Communication*, Vol. COM-28, (April 1980), 468-488.