Computing in fault tolerance broadcast networks

Ilan Newman
Dept. of CS,
University of Haifa, Haifa, Israel
ilan@cs.haifa.ac.il

Abstract

We consider a fault tolerance broadcast network of n processors each holding one bit of information. The goal is to compute a given Boolean function on the n bits. In each step, a processor may broadcast one bit of information. Each listening processor receives the bit that was broadcasted with error probability bounded by a fixed constant ϵ. The errors in different steps, as well as for different receiving processors in the same step, are mutually independent. The protocols that are considered in this model are oblivious protocols: At each step, the processors that broadcast are fixed in advanced and independent of the input and the outcome of previous steps. The primal complexity measure in this model is the total number of broadcasts that is performed by the protocol.

We present here the first linear complexity protocols for several classes of Boolean functions, including the OR function, functions that have $O(1)$-minterm (maxterm) size, functions that have linear size AC_0 formulae and some other functions. This answer an open question of Yao [22], considering this fault tolerance model of El Gamal [2] and Gallager [8].

1. Introduction

We consider computations in a noisy broadcast network under the standard (adversarial) *fault tolerance* model. In a fault tolerance scenario the outcome of any single operation is erroneous with probability bounded by some constant ϵ. It is also assumed that errors for different operations are independent. Given a complex algorithm, one often wishes to get a noise-immune algorithm, namely one that will end with an overall constant error probability. If the outcome of each operation is Boolean, and there are total of t operations, then repeating each operation for $O(\log t)$ times and taking majority as the outcome, typically results in a noise-immune algorithm at the cost of $O(t \log t)$ operations. It would be desirable to construct noise-immune algorithms paying less overhead. This fault tolerance scenario was extensively studied for different computational models. Such work includes [21, 1, 14, 7, 16] for Boolean circuits with noisy gates, [5, 16, 3, 10] for noisy decision trees, [17, 18, 15] for the communication complexity model and others [20, 12, 6, 19]. It turns out that sometimes it is impossible to do better than the overhead factor of $O(\log t)$ and sometimes there is a non-trivial better way.

El Gamal formulated the following *noisy* broadcast network model, typical for some radio networks. The system is composed of n processors using broadcasts as the mean of communication. At each step of a protocol, each processor may broadcast a bit, while every processor receives all the bits transmitted knowing the source of each one. Each bit received, at each processor, might be erroneous (in its value) with a probability bounded form above by an a priori known constant ϵ. Furthermore, it is assumed that the errors for different broadcasts, as well as for different receivers in the same broadcast, are independent. At the beginning of the operation, each processor has one bit. The goal is to compute a given Boolean function on all the bits. The complexity of the protocol is the total number of broadcasts. The protocols that are considered are *oblivious*, i.e. whether at a step i processor P_j broadcasts or not, may not depend on its initial bit and/or on what he has received so far. It may only depend on n, i and j. The content of the transmission may depend on the inputs. The reason for considering oblivious protocols is both practical, as a model of certain radio networks, as well as the natural way to avoid situations in which information is inferred from whether a certain P_i has broadcasted or not, rather then from the content. Note that for non-oblivious protocol, all the bits can be recovered in a single round of $O(n)$ broadcasts in which each processor that holds a '1' broadcasts.

Gallager [8], showed that the parity functions can be computed in $O(n \log \log n)$ broadcasts which was the first improvement on the trivial $O(n \log n)$ solution. His protocol ends with all processor knowing all bits, hence implying the same bound for every Boolean function. Yao, [22] posed it as an open question whether the OR function (or other non-trivial functions) can be computed in $O(n)$ broadcasts. To date, Gallager's $O(n \log \log n)$ protocol is the best known to end with all processors knowing all input bits. For the specific OR function, Feige and Kilian [4] constructed an $O(n \log^* n)$ protocol that has the additional feature of taking $O(\log^* n)$ rounds. Kushilevitz and Mansour [11] constructed a linear complexity protocol for computing threshold functions, but in a considerably more restrictive model, which we call here the *statistical* model. In their model it is assumed that there is fixed constant ϵ (known or unknown to the protocol) so that for each pair of processors (P_i, P_j), the error of P_i when receiving a transmission of P_j is **exactly** ϵ. In particular, in such a model, the parameter ϵ can be estimated, even if unknown, to any predefined precision, by sampling. A protocol, designed for such model for the fixed parameter ϵ, is not required to work for the situation where ϵ decreases, and in particular if it becomes 0.

In the fault tolerance model, in contrast with the statistical model, the error distribution is not known! There is only a guarantee that the error per operation is *at most* ϵ and that errors are independent for different operations. Any adversarial probability distribution that meets those limitation is a legitimate one. Hence, a fault tolerant algorithm designed for a parameter ϵ should certainly work, within its defined complexity and error bound, for the situation in which the actual error is bounded by any $\delta < \epsilon$, and in particular, for the errorless case.

We answer in the affirmative Yao's question: We construct here several $O(n)$ fault-tolerant protocols in the standard noisy broadcast model for various classes of Boolean functions including the OR, AND, functions that have $O(1)$ size 1-witnesses (0-witnesses), functions that have linear size AC_0 formulae and some other functions. Thus, we give the first linear complexity protocols for both the OR functions, and some other interesting non-trivial classes of Boolean functions. We also show how to find the whole input word using $O(n \log r)$ broadcasts, provided that the number of 1's in the input is at most r. Note that for r such that $\log r = o(\log \log n)$ this is better than Gallager's bound. We also treat briefly the issue of the number of rounds needed for the OR function. We construct a $O(n)$ complexity fault-tolerant decision tree algorithm of $O(\log^* n)$ rounds and comment on how to implement it as a $O(\log^* n)$-rounds protocol in the broadcast model.

Our results are based on three main ingredients. The most important one is a reduction of the task of computing Boolean functions in the broadcast network model to certain efficient computation in the fault tolerance decision tree model. In order to implement fault-tolerant decision tree algorithms in the broadcast model we introduce a property that measures the adaptiveness of such algorithms, denoted here as *small parallel time*. We construct new algorithms for some classes of Boolean functions for the fault tolerance decision tree model. In particular we construct two very simple linear algorithms for the OR, with one being simultaneously of total linear complexity and $O(\log^* n)$ rounds ($O(\log^* n)$ parallel time). The second ingredient is a simple application of some ideas of parallel computing. The third is, similar to the work of [4] (and unlike the algorithm of [11]), the ability to produce witnesses for the OR function.

The rest of the paper is organized as follows: In Section 2.1 we define the model and observe some basic properties. In Section 2.2 we define the fault tolerance decision tree model and present an $O(n)$ algorithm for the OR, as well as some other functions. In Section 3 we describe a linear time protocol for the OR function in the fault tolerance broadcast model. In Section 4 we discuss the general issue of simulating fault tolerance decision tree algorithms in the broadcast model. We then develop algorithms for several Boolean functions in the fault tolerance decision tree model which can be simulated in broadcast model. In particular, we construct linear complexity fault-tolerant protocols for functions that have $O(1)$-size 1-witnesses (0-witnesses) and for functions that have linear size AC_0 formulae. Then, in Section 5 we discuss the all-bit word problem in which the outcome should be that every processors knows the whole input word. We show that if the number of 1's in the input is very low, the all-bit problem can be solved in less than the Gallager bound. In Section 6 we discuss briefly algorithms for OR that are very efficient in the total number of rounds, in both the decision tree model and the broadcast model. Finally, in Section 7 we discuss some open problems.

2. Preliminaries

2.1. The model

Let $P_1, ..., P_n$ be n processors. An ϵ-fault tolerant protocol in the fault tolerance broadcast model is a sequence of m rounds; in every round $j = 1, ..., m$, a certain subset $S_j \subseteq [n]$ of the processors broadcast, one bit each. For each processor $P \in S_j$, every processor Q, $Q \neq P$ receives a random variable $X^j_{P,Q}$ that equals the bit that was broadcasted by P with probability at least $1 - \epsilon$. In addition, the variables $X^j_{P,Q}, j = 1, ..., m,\ P \in S_j,\ Q \in [n] - \{P\}$ are independent. Furthermore, the subset S_j, namely the identity of the processors that broadcasts at round j is predefined in the protocol and is independent of the input or previous information that processors might obtain in the rounds previous to j. Note that the value of the bit that is transmitted by P at round j might depend on the input as well as the previous information received by P. This restriction of being *oblivious* is to avoid trivialities such as the following one round protocol: Each processor broadcasts if and only if it has a '1'. Then all input bits become known to each processor, regardless of any error, just by detecting which processors have broadcasted.

The complexity of an oblivious protocol as above is the total number of broadcasts, namely $\Sigma_{j=1}^{m}|S_j|$. Another complexity parameter that is of (secondary) interest is m, the number of rounds of the protocol. A protocol is said to compute a (partial) function $f()$ if for any allowed input it ends with a value f_i at each processor $i = 1, ..., n$, such that $Prob[\exists i\ f() \neq f_i] \leq 1/3$.

Note, as already discussed in Section 1, that ϵ, the bound on the error probability is a priori known. It is a perfectly legitimate for an alleged adversary to corrupt each, say, odd round j with probability $\delta_j < \epsilon$ while letting each even round to result in an error-less reception (or alternatively, decide for each processor r and each round j to corrupt X_r at the j round with probability $p_{r,j} < \epsilon$). Obviously, any protocol can be serialized keeping the same number of broadcasts. We will some times do this in order to refer to some particular order of the broadcast within a round. In this case we talk about steps in which a single processor broadcasts.

Finally since simple amplification can be done in this model, the exact value of ϵ is not important. Also, for the same reason we could make the final error probability bound to be any constant (instead of $1/3$) at a linear cost.

An important 'gadget' that we recurrently use is the following computational task. Suppose that a fixed known processor P holds a word $w \in \{0,1\}^k$ that he wishes to make known to every other processor. We denote this task as the 'faithful distribution' of w. The following simple observation states that 'faithful distribution' can be done efficiently.

Observation 2.1 *Faithful distribution of $w \in \{0,1\}^k$ to n processors takes $O(max\{k, \log n\})$ broadcasts. As a result each processor knows w correctly with probability $1 - 1/poly(n)$.*

Proof: Suppose first that $k \geq 16 \log n$. Let us fix a linear size error correcting code that corrects up to $1/3$-fraction of errors (the fact that such codes exists is widely known see e.g [13]). P_1 encodes w using the above code into a word $c(w)$ of size $C(k) = O(k)$ and broadcasts each bit of $c(w)$. Each $P_i,\ i = 1, ..., n$ decodes w from the received word. P_i erroneously decodes w only if more than $C(k)/3$ errors occurred. By Chernoff, this will occur with probability at most $exp(-2(1/3 - \epsilon)^2 C(k))$. For $\epsilon < 1/12$, this would be less than $1/n^2$ (or below any $1/poly(n)$ for suitable ϵ. This implies that with probability at least $1 - 1/n$ all processors have decoded w correctly. If $k < 16 \log n$ we append w to itself $16 \log n/k$ times, to form a word of size exactly $16 \log n$. We then faithfully distribute it using $O(\log n)$ broadcasts.

2.2. Noisy Boolean Decision-Trees

A major ingredient in what follows are algorithms for fault tolerance Boolean decision-trees. Several such models were discussed in the literature. The relevant one to this work is the seminal work of of Feige et al. [5]. In this model, a given Boolean function in n variable is to be computed on unknown input by queries to the variables. As a standard fault tolerance model, for each variable queried, an erroneous answer is received with probability bounded by an a priori bound ϵ and independent of any other queries or answers. Hence, an algorithm is a decision tree. In each step a variable is queried, then, based on the answer a next variable is being queried and so on, until a final decision is made. The complexity of such algorithm is the maximum number of queries for the worst case input. Namely, it is the depth of the tree. A tree $\mathcal{T}$ computes a (partial) function $f()$ if, for every input, its error probability is bounded by $1/3$. Again, as standard amplification is possible, we may replace the $1/3$, as well as ϵ, with any other constants. We say that $\mathcal{T}$ is an ϵ-fault tolerant algorithm if, for input error bounded by ϵ, the output error is also guaranteed to be bounded by ϵ.

An important parameter of complexity, although non-standard for decision trees, is the *amount of adaptiveness,* which can be measured by the 'parallel time' of the algorithm. Namely, for each input, the algorithm defines a certain path in the tree, in which a sequence of variables are probed one after the other. However, in many cases, this order is arbitrary in the sense that the query at step j does not depend on the answer at step $j - 1$ for some computational path. In such a case, the queries of step $j - 1$ and step j could have been asked in parallel. If the algorithm is presented as a decision tree, then there is no explicit parallelism. However, as we care about this extra parameter, the algorithms will be described in terms of rounds: In each round a (multi)set of variables are queried. The complexity of the algorithm is the total number of queries, while the parallel time is the number of rounds for the worst case run. Feige et al. proved (among other things):

Theorem 1 *[5] There is a fault tolerance decision tree for the OR function on n variables of complexity $O(n)$ and parallel time $O(\log n)$.*

We present an explicit algorithm for the above (with a much simpler analysis than of [5]). In Section 6 we construct a variant of this algorithm that is simultaneously of $O(n)$ total complexity, while it takes only $O(\log^* n)$ rounds. The following is, however, simpler and will turn more 'handy' to simulate as a protocol for the noisy broadcast model.

Observation 2.2 *There is an $1/15$-fault tolerance decision tree $\mathcal{T}$, that for any input $x \in \{0,1\}^n$ outputs an index*

$i \in [n]$ which is the minimal for which $x_i = 1$, if such i exists, or an arbitrary i if $OR(x) = 0$. The complexity of $\mathcal{T}$ is $O(n)$ while its parallel time is $O(\log n)$.

Proof: Let $\epsilon < 1/15$ and $n = 4m$. $[n]$ is partitioned into m disjoint subsets of size 4, denoted by blocks. The algorithm $\mathcal{T}$ will be recursive. Every variable $x_i,\ i = 1, ..., n$, is queried 3 times, and a majority of the three answers is taken as its temporary value z_i. Then, among the 4 values obtained in each block, an index i is chosen to be the minimal for which the value $z_i = 1$, if any of the values is 1, or an arbitrary i in the block, if all values are 0's.

Let $I \subseteq [n]$ be the set of m values thus chosen. Then $\mathcal{T}$ is applied recursively on I for 3 independent times. On each of those 3 phases $\mathcal{T}[I]$ ends with an index i as a result. The result of $\mathcal{T}$ is now set to be j if j is obtained as an answer at least twice out of the 3 independent runs on I, or arbitrary if there is no such j.

Clearly the total complexity of $\mathcal{T}$ on n variables, denoted by $a(n, \epsilon)$ is bounded by $a(n, \epsilon) \leq 3n + 3a(n/4, \epsilon)$ which implies that $a(n, \epsilon) \leq 12n$. It is also clear that the parallel time is $O(\log n)$.

Let ϵ_n be the error of the algorithm on n variables. We show inductively that $\epsilon_n \leq \epsilon$ (which is easily checked for $n = 4$). To analyze the error probability note that $\mathcal{T}$ can't err if $OR(x_1, ..., x_n) = 0$. Assume then that $OR(x_1, ..., x_n) = 1$ and let i be the minimum index for which $x_i = 1$. Then $\mathcal{T}$ errs if $i \notin I$, or if $\mathcal{T}$ errs in at least twice out of the 3 independent runs on I. The probability of the first event is clearly bounded by $4 \cdot Prob(z \neq x)$, where x is any variable and z is its corresponding intermediate value, (as for i not to get into I, at least one of the values in the block must have been wrong). For each individual i, $Prob(z_i \neq x_i) \leq 3 \cdot \epsilon^2$, hence the probability for the first event is bounded by $12\epsilon^2$. The probability of the second event is bounded by $3\epsilon_m^2 \leq 3\epsilon^2$ by induction. Hence $\epsilon_n \leq 15\epsilon^2 \leq \epsilon$ (as $\epsilon \leq 1/15$). ■

Remarks:

1. Another feature of the protocol $\mathcal{T}$ above which will be handy is the following: As $\mathcal{T}$ runs in some fixed $t = O(\log n)$ rounds, we may describe $\mathcal{T}$ as a sequence $S_1, ..., S_t \subseteq [n]$ of queries, namely S_i containing those variables (with multiplicity) that are queried in the ith round. Note that S_i might depend on the answers to the previous queries. However, note that the sizes $|S_i|,\ i = 1, ..., t$, are fixed and independent of the input or the current run.

2. Once the index r above is found, x_r can be queried for $O(\log n)$ times, enough to verify that $x_r = 1$ with probability at least $1 - \frac{1}{n^2}$, hence producing an 'effective' witness for the case the function is '1'.

3. A Protocol for the OR in the fault tolerance broadcast model

Our aim is to simulate the algorithm $\mathcal{T}$ of Observation 2.2 and find the first index of a variable that is '1' (if such exists) in linear number of broadcasts. It will now become evident why the 'limited adaptiveness' is crucial. We note that the algorithm of [4] which takes $O(n \log^* n)$ broadcasts and $O(\log^* n)$ rounds, also finds the first processor that holds a '1' (if exists). Here we show how to construct such protocol with $O(n)$ broadcasts. In Section 6 we indicate how to obtain a simultaneous $O(n)$ complexity, $O(\log^* n)$-rounds protocol.

Theorem 2 *Let $P_1, ..., P_n$ be n processors, each holding a Boolean value $b_i,\ i = 1, ..., n$, and an extra processor P_0. There is a $1/30$-fault tolerant protocol, $\mathcal{A}$, of complexity $O(n)$, that ends with P_0 holding an index $r \in [n]$. If $OR(b_1, ..., b_n) = 1$ then r is the minimal index for which $b_r = 1$, otherwise r is arbitrary.*

Proof: Let $\mathcal{T}$ be the algorithm of Observation 2.2. Following the remark at the end of Section 2.2, let $S_1, ..., S_t \subseteq [n]$, be the multisets of variables that are queried at round i, $i = 1, ..., t = O(\log n)$ for a given run. Recall that S_i might depend on the answers of previous queries to variables in $S_{i-1}, ..., S_1$, however, its size $|S_i|$ is known and fixed for all runs. Also, note that $\Sigma_{i=1}^{t} |S_i| \leq 12n$ (here $|S_i|$ is a multiset of the variables queried in the round, hence a variable that is queried several times counts that many times in the multiset). Let $\epsilon = 1/30$ be the bound on the error probability in each broadcast.

We first describe the top level idea for P_0 to simulate $\mathcal{T}$ by a *non-oblivious* protocol. We then explain the details of the oblivious protocol. To simulate the first round, P_0 needs to query the variables in S_1. As $\mathcal{T}$ is known to all processors, this can be done by letting each $P_i,\ i \in S_1$ broadcast its value. At this point P_0 successfully have simulated the first round of $\mathcal{T}$. However, at this point, to carry out the second round in a similar way, each P_i needs to know the values that P_0 obtained in the first round of queries. This poses no real problem as P_0 can faithfully distribute his word of answers using $O(max\{|S_1|, \log n\})$ broadcasts, as described in Section 2.1. At this point the second round of $\mathcal{T}$ can be simulated as before and so on. Each set of queries, S_i, is thus done by $|S_i|$ broadcasts followed by a faithful distribution of the answers which takes $O(|S_i|)$ (if $|S_i|$ is large enough). Hence, this would total to $O(\Sigma |S_i|) = O(n)$, providing we can take care of two points: The first is the non-oblivious behavior; to pay only $|S_i|$ broadcasts for the queries in S_i, only those processors that are in S_i should broadcast. As S_i may depend on previous answers this is not oblivious. We deal with this issue later. The second point, which is simpler, is the inaccuracy of the analysis due to the fact that $|S_i|$ might be very small for some i's. In this case, the faithful distribution gadget of Section 2.1 takes $\theta(\log n)$. To account for this we make use of the limited adaptiveness; as there are only $O(\log n)$ rounds in $\mathcal{T}$, all rounds of less than $12 \log n$ queries contribute a total complexity of $O(\log^2 n)$ in all the faithful distributions of their answers. All other rounds, namely those for which $|S_i| \geq 12 \log n$ contribute at most $O(\Sigma_{i=1}^{t} |S_i|) = O(n)$ complexity, hence the total complexity is linear even though the faithful distribution of some rounds may cost $\theta(\log n)$.

It is important to note the following: although S_i might depend on the outcome of previous queries, P_0 does not specify S_i itself, but rather just the answers to the queries in S_{i-1}. As the tree $\mathcal{T}$ is known to all processors, this is sufficient to uniquely determine S_i. Hence, after the $(i-1)$th round was simulated, P_0 needs to faithfully distribute only $|S_{i-1}|$ bits of information in order to make S_i known to all processors.

We are now back to the problem of obliviousness. We assume in what follows that we have an extra $12n$ processors, $H_1, ..., H_{12n}$, denoted as helpers. The helpers do not have any values at the beginning of the protocol. Clearly, an oblivious protocol using such $12n$ extra helpers can be simulated by an oblivious protocol on the original $n+1$ processors paying an overhead of a 12-factor to the total complexity (by letting each 'real' processor simulate 12 helpers - we avoid further details on this here.)

The idea of how to get rid of the non-obliviousness is now simple. We start by letting each processor $P_i,\ i = 1, ..., n$ broadcast its bit. As a result each helper H_j contains a 'copy' $Z_j(i)$ of the bit of P_i for every i. From this point on, the original processors $P_1, ..., P_n$ do not participate in the protocol, only P_0 and the helpers are responsible for the rest of the work. Helper $H_j,\ j = 1, ..., 12n$, will be used just once during the whole protocol - to simulate the answer to the jth query done by $\mathcal{T}$. Now to simulate $\mathcal{T}$ in an oblivious way, note that the non-oblivious protocol above works in t phases, each contains two rounds. The first round is a 'query-round' in which a 'set of queries' of $\mathcal{T}$ is simulated by letting the processors in S_i broadcast their bits. The second round is a 'faithful-distribution' round in which P_0 faithfully distributes the answers. The query rounds are done by the helpers: If the qth query in $\mathcal{T}$ is to $b_j, j \in S_i$, then in the ith phase, at the query round, H_q broadcasts $Z_q(j)$. Note that the identity of H_q is known in advance ! The value it needs to broadcast, $Z_q(j)$, namely as determined by j, becomes known prior to the ith phase by the faithful-distribution rounds. Interleaving with the query rounds are the rounds of faithful distribution, in which P_0 is the one that broadcasts. As the size of each query set, in each round of $\mathcal{T}$, is known and fixed in advance, the number of actual steps in each faithful-distribution round is known in advance. Thus, it is known in advance in which step P_0 broadcasts and in which step $H_q,\ q = 1, ..., 12n$ does. This shows that the whole protocol is oblivious as required.

Finally, note that from the view point of P_0, a broadcast of $Z_j(i)$, taken as an answer to a query of x_i, is identical to the true value of x_i with error probability at most 2ϵ (as an error may occur either when the helper receives his copy, or when P_0 receives his). Moreover the errors in different 'query broadcasts' are independent. Hence, if there was no other source of error, the error probability would be bounded by that of $\mathcal{T}$, for 2ϵ input error. We assume that $\epsilon \leq 1/30$ and that $\mathcal{T}$ is a $1/15$-fault tolerant tree, hence, the output error would be bounded by $1/15$. There is another source of error though, which is the event that in some of the faithful distributions, one of the processors decodes erroneously the information. However, the error in the decoding at each round (in any processor) can be made smaller than $1/n$ and hence all rounds of faithful distribution are correct with probability at least $1 - O(\log n/n)$. Thus the total error is bounded by $1/15 + o(1)$. ■

Remark: We note that using helpers in the proof above works as the errors resulted from the broadcasts of different helpers are independent. However, this is true as P_0 is the sole one to use this information. If more than one listener is using the information of any one helper, all listeners might get the wrong value with constant probability, as the helper might get a faulty value.

Protocol $\mathcal{A}$, above, does not fully solve the OR function as only P_0 knows the answer r. It is quite simple to construct from it a protocol for the OR, which in addition, produces a 1-witness in case the answer is '1'.

Theorem 3 *There is an $O(n)$-complexity protocol that computes the OR function in the fault-tolerance broadcast model. In addition, for every input for which the answer is '1', it produces an index r for which $x_r = 1$.*

Proof: Let P_0 be an extra processor. First, protocol $\mathcal{A}$, of Theorem 2, is performed, which ends with P_0 knowing an index r as required. Now, P_0 faithfully distribute the index it gets as an outcome of $\mathcal{A}$ and then let the next $\theta(\log n)$ helpers broadcast the corresponding value. Each P_i, taking the majority of those latter $\theta(\log n)$ values, obviously gets the correct answer with error probability that can be made smaller than $1/n^3$. Hence, the total error, given that $\mathcal{A}$ ended correctly, is bounded by $1/n^2$. Finally, the actions of P_0 can be simulated by P_1, which eliminates the need of the extra processor. ■

Clearly, Theorem 3 imply the same result for the AND function.

4. Protocols for other Functions

The simulation of a decision tree algorithm by an oblivious broadcast protocol is quite general;

Theorem 4 *Let $f : \{0,1\}^n \longrightarrow \{0,1\}$ be a Boolean function for which there is a fault tolerant decision tree of complexity $C(n)$ and parallel time $t \leq \frac{C(n)}{\log n}$ then f can be computed by a fault tolerant broadcast protocol of complexity $O(C(n)^2/n)$.*

Proof of Theorem 4 (sketch): Let $\mathcal{T}$ be the fault tolerant decision tree for f of complexity $C(n)$ and parallel time $t \leq C(n)/\log n$. We first transform $\mathcal{T}$ into a decision tree $\mathcal{T}'$ of complexity $O(C(n))$, in which exactly $12\log n$ queries are being made in every round. Then we simulate $\mathcal{T}'$ by an oblivious protocol along the lines of the proof of Theorem 2.

To do this, we consider a round of $\mathcal{T}$ in which more than $12\log n$ queries are asked, denote as a 'large' round. We partition the queries of such round into subsets of size exactly $12\log n$, possibly except for one, which is of size at most $12\log n$. We then ask the queries in those subsets in consecutive rounds. At this point get an algorithm $\mathcal{T}''$ in which in every round there are either $12\log n$ queries or less. The number of queries in $\mathcal{T}''$ is identical to that of $\mathcal{T}$. The parallel time of $\mathcal{T}''$ is $O(C(n)/\log n)$. To see this, note that in $\mathcal{T}$ there are at most $C(n)/(12\log n)$ large rounds. Each large round results in at most one subsets of size smaller than $12\log n$. Hence, those new small sets contribute $O(C(n)/12\log n)$ to the parallel time of $\mathcal{T}''$. All other subsets that are resulted by large rounds, are of size exactly $12\log n$. As the total complexity of $\mathcal{T}''$ is $C(n)$ there can't be more than $C(n)/12\log n$ such subsets, each adding one round. Hence the total parallel time of $\mathcal{T}''$ is at most that of $\mathcal{T}$ plus $O(C(n)/12\log n)$.

Now we transform $\mathcal{T}''$ into $\mathcal{T}'$: we add arbitrary queries to every round in which less than $12\log n$ queries are made, to make it of size exactly $12\log n$. The parallel time remains $O(C(n)/\log n)$ while each round now is of size exactly $12\log n$. This also implies that the total complexity is $O(C(n))$.

At this point we can just simulate $\mathcal{T}'$ exactly as done in Theorem 2, using $C(n)$ helpers. Each round of $\mathcal{T}$ is of $12\log n$ queries, hence, the total overhead incurred by the faithful distribution of answers is linear in $O(C(n))$. Finally, to simulate the $C(n)$ helpers by the original n processors, an overhead of $C(n)/n$ must be payed (as each processor simulates $C(n)/n$ helpers), which concludes the proof. ■

We describe here several applications of the tools that we have seen so far.

The first application is by composing algorithms in the fault tolerance decision tree model, to obtain efficient algorithm for compositions of functions. Let $f : \{0,1\}^n \longrightarrow \{0,1\}$ be computed by an ϵ-fault tolerance decision tree of complexity C_f and parallel time T_f, let $g_1, ..., g_n : \{0,1\}^m \longrightarrow \{0,1\}$ be Boolean functions each having an ϵ-fault tolerance decision tree of complexity C_i and parallel time T_i, $i = 1, ..., n$, respectively. Then the composition $f(g_1, ..., g_n) : \{0,1\}^{n\cdot m} \longrightarrow \{0,1\}$ can be computed by an ϵ-fault tolerant decision tree of complexity $C_f \cdot max_{i=1}^n C_i$ and parallel time $T_f \cdot max_{i=1}^n T_i$. This is due to the fact that one can 'compose' the algorithm for f with those for the $g's$ in the straight forwards way (as the assumption on the input error bound is equal to the assumption on the output error bound). Namely, the algorithm for $f(y_1, ..., y_n)$ is applied and every time there is a query to y_i, a new run of the algorithm to g_i is made. Hence, composition does not require any amplification which would normally be required in order to compensate for the large number of input bits, or intermediate functions.

As an example let $[n]$ be partitioned into $m = \sqrt{n}$ disjoint 'blocks' $B_1, ..., B_m$, each of size m. We index n Boolean variables by $[m] \times [m]$ and let $f = \wedge_{j=1}^m \vee_{i \in B_j} x_{i,j}$. To compute f by a $1/15$-faulty decision tree we compute the top AND in $O(m)$ queries using Theorem 3, where each query is a query to the value of an OR_m function. Each such query can again be computed with error bound $1/15$ in $O(m)$ simple queries to the variables (that are assumed to have error bound of $1/15$).

In particular this also implies,

Observation 4.1 *Let $f : \{0,1\}^n \longrightarrow \{0,1\}$ be a Boolean function that has an AC_0 formula of size s. Namely, f can be computed by an unbounded fan-in Boolean formula with s leaves and constant depth. Then f has an $O(s)$ complexity fault tolerance decision tree of $O(poly(\log n))$ parallel time.*

Corrolary 4.1 *Let f be computed by s-size AC_0 formula, then f can be computed by a fault tolerance broadcast protocol of complexity $O(s^2/n)$. In particular, if $s = O(n)$, f can be computed by a linear complexity, fault tolerance protocol.*

proof Sketch:(of Observation 4.1) Immediate form the fact that OR, AND can be computed in an ϵ-fault tolerant way in linear time, and the fact that f can be computed by computing the OR, AND of its subformulae, where each is computed recursively (namely using composition as in the discussion above). As the total number of appearances of variables at the bottom level (of the formula, which is also the bottom level of the recursion) is s the result follows. We leave out further details in this abstract. ■

Another class of functions that can efficiently be computed in the fault tolerance broadcast model is the class of functions that have $O(1)$ size minterms or maxterms. King and Kenyon, [10], proved that any function of $O(r)$ size minterms (not necessarily monotone) can be computed by a fault tolerance $O(n \log r)$ size decision tree. However, their algorithm might have linear parallel time. We show;

Theorem 5 *Let f be a Boolean function whose minterm size is at most $r = O(1)$, then f can be computed by a fault tolerance decision tree of complexity $O(n)$ and parallel time $O(\log^2 n)$.*

The algorithm that we construct is inspired by the ideas of [10], however, it is quite different and of very different and simpler analysis. The proof will follow from Theorem 6 for monotone functions.

As a corollary of Theorem 5 and Theorem 4 we get

Corrolary 4.2 *Let f be a Boolean function whose minterm size is at most $r = O(1)$, then f can be computed by an $O(n)$ fault tolerant broadcast protocol.* ■

Theorem 6 *Let f be a monotone Boolean function whose minterm size is at most $r = O(1)$, then f can be computed by a fault tolerance decision tree of complexity $O(n)$ and parallel time $O(\log^2 n)$.*

Proof of Theorem 6 (sketch): Let $f : \{0,1\}^n \longrightarrow \{0,1\}$ be a monotone function whose minterm size is bounded by some $r = O(1)$. We represent the collection of minterms of f by an hypergraph $H_f = ([n], E)$ in which every hyperedge $e \in E$ is a minterm. A matching in H_f is a collection of pairwise disjoint edges.

The Claim is proved by induction on r. For $r = 1$ this is just the OR. Also recall that the OR produces a witness for which we can verify a '1' answer in $O(\log n)$ broadcasts and error probability at most $1/n^2$.

We first consider a simple case in which the maximum matching in H_f is of size at most $n/\log n$. In this case there is a set $C \subseteq [n]$ of size less than $r \cdot n/\log n = O(n/\log n)$ that covers E in the following sense: For every $e \in E$ there is a vertex $i \in C$ for which $i \in e$ (simply by taking all vertices in all edges of a maximal matching). To evaluate f we first query each variable $x_i,\ i \in C$ for $\Theta(\log n)$ times, enough to know its value with error probability less than $1/n^2$. This is done in one round at total complexity $O(n)$. Now we may restrict ourselves to the sub function $f_1 = f|_{\bar{C}}$ that is defined on those variables outside C. As each minterm ($e \in E$) is covered by C, it follows that f_1 has minterms of size bounded by $r-1$. Applying the induction hypothesis ends the proof in this case.

Assume now that H_f contains a matching $M \subseteq E$ of size exactly $n/\log n$ (if the size is larger we take a partial matching). The algorithm will work as follows. For each variable x_i it queries, it will keep a value $val(x_i)$ that will be the difference between the number of '1' answers and the number of '0' answers on x_i. As soon as $val(x_i) = -1$ for some x_i, we decide that $x_i = 0$ and look at the restricted function where x_i is substituted with '0'. The important feature is the way the algorithm decide on the next set of variables to query. First, it finds a matching M of size $m = n/\log n$ in H_f. Clearly then, f can be written as $f_M \vee f'$ where $f_M(x) = \vee_{e \in M}(e(x))$ and f' is the function whose minterms are in $H_f - M$. The algorithm applies the OR algorithm on the m disjoint minterms to evaluate f_M for three independent times and takes majority of the results. To do this, every time that a value of a minterm has to be evaluated, all variables in it are queried for $O(\theta(\log r))$ times so that the answer to the whole minterm would be evaluated with ϵ-error, as needed for the OR. Hence, a round of computing the OR on a matching is done in $O(mr \log r)$ queries. As a result we compute the OR with error probability bounded by $3\epsilon^2$. If f_M evaluates to 1 we are done and stop. If it evaluates to 0, we prove that with high probability at least $m/2$ of the variables that had nonnegative value before, reach a negative value, and thus are set to '0' and deleted from the function. If this almost sure event does not occur we answer '0' and stop.

Thus, at each round, we either end (if f_M evaluates to 1), or we delete $m/2 = \Omega(n/\log n)$ variables from f. Hence, this can proceed for at most $O(\log n)$ rounds, after which either the algorithm ends, or f is reduced to the $r-1$ case as explained before. The total complexity in such $O(\log n)$ rounds is $O(\log n \cdot mr \log r) = O(nr \log r)$, which is linear for $r = O(1)$. Hence the total complexity (including the descent into $r = O(1)$ recursion levels) is linear.

To analyze the error probability we assume inductively (on r) that the error probability is bounded by $r^2 \cdot \epsilon/(1-\epsilon)$. Note that the error for a '0' input is negligible, as the algorithm answers '1' only if a witness is found, namely a minterm for which the majority of at least $\Omega(\log n)$ queries to its value is '1'. As previously noted, this would have error at most $1/n^2$. Thus the maximum error is for 1-inputs. Another fact to note is that if $x_i = 1$, $Prob(val(x_i) = -1) = \epsilon/(1-\epsilon)$ which can be proved by considering $val(x_i)$ as a random walk on the integers (see e.g [9] Chapt. 3).

Let $x \in f^{-1}(1)$ and let $M_1, M_2, ...$ be the matching that the algorithm encounters on x for some run, and before f is reduced to some function f_1 whose minterms are of size at most $r-1$. Let $V_i = \cup_i \cup_{e \in M_i} e$. Namely, V_i are all the variables that are contained in any of the minterms belonging to any of the matchings $M_1, ..., M_i$. At each stage of the algorithm, some minterms are deleted; a minterm P may be deleted if $P \in M_i$ and M_i was evaluated to 0, or if some $x_j \in P$ was deleted at some round k in which $val(x_j)$ first became -1. As $f(x) = 1$, x satisfies one or more minterms. Hence, for the algorithm to evaluate f to 0 on x, there are three possible events: (a) all the satisfied minterms must have been deleted. (b) f was reduced to a function f_1 whose minterms are of size at most $r-1$, $x \in f_1^{-1}(1)$ and the algorithm errs on f_1. Or (c) a matching

M_i for which $f_{M_i}(x) = 0$ was evaluated to 0, but less than $m/2$ of the variable of positive values after round $i-1$ became negative after round i (recall that we evaluate f as 0 in this case).

Now, in case (c), as $f_{M_i}(x) = 0$, every one of the m disjoint minterms in M_i must contains at least one variable that is 0. For each such variable, being positive (after any number of queries) occurs with probability bounded by 2ϵ, hence by Chernoff, at list $m/2 = \Omega(n/logn)$ variables are negative with extremely high probability. Case (b) is reduced to the case of minterm size of $r-1$, which would be bounded inductively by $(r-1)^2 \cdot \epsilon/(1-\epsilon)$. Case (a) might occur in two different ways: Let P be the first minterm that is satisfied by x and was deleted. Either $P \in M_i$ and was deleted as M_i was evaluated to 0. This occurs with probability bounded by $3\epsilon^2$ which is the error probability of the OR on M_i. Or P is deleted as some of its variables became negative. The probability for such a variable to become negative is bounded by $\epsilon/(1-\epsilon)$ as noted above. As there are r variables in P this contributes at most $r\epsilon/(1-\epsilon)$. Hence the sum over those three cases results in at most $r^2\epsilon/(1-\epsilon)$ as required.

Finally, to see that the parallel time is bounded as needed, note that each round that is defined by a large matching M causes (with high probability) the deletion of $\Omega(n/logn)$ vertices, hence there are $O(\log n)$ such rounds. The computation in each round is an OR of minterms which is of $O(\log n)$ rounds. Then, there might be a multiplication of the above r times by the inductive process. Hence the parallel time is $O(r \cdot \log^2)$. We defer further details to the full version. ■

Proof of Theorem 5: To prove the Theorem for a non-monotone function f we just need to replace every appearance of $\bar{x_i}$ (the negation of x_i) in a minterm, with a new variable z_i. Then we get a new monotone function f' on $2n$ variables, for which f is a sub function. We then evaluate f' on the new variables while treating x_i, z_i as independent variables. The Theorem now follows form the monotone case. ■

5. Recovering the whole input word

As mentioned in Section 1, Gallager's protocol computes the full input word in $O(n \log\log n)$ broadcasts. Theorem 3 implies that the whole input word can be computed in $O(n \cdot r \log r)$ broadcasts, provided that there are at most r 1's in it (just by finding the OR $r \log r$ times, spending $O(n \log r)$ broadcasts per witness). Can we do better? We get.

Theorem 7 *For any r, there is a protocol of complexity $O(n \log r)$ that for every input word w, it either computes the whole word w, or it finds at least r positions in which w is '1'.*

Proof Sketch: Note that for r such that $\log r = \Omega(\log\log n)$ the whole word can be computed in $O(n \log\log n)$ by Gallager's result. We may assume in the following that $\log r = o(\log\log n)$. We partition $P_1, ..., P_n$ into $m = r^2$ blocks, denoted $B_1, ..., B_m$. Let $Z_i,\ i = 1, ..., m$, be the *true* value the OR of the input bits, in the ith block, respectively. Let $P'_i,\ i = 1, ..., m$, be the first processor in each block.

The algorithm is composed of two main phases: In the first phase, we want to recover the word $Z = (Z_1, ..., Z_m)$. To do this we compute the OR on each block $\theta(\log r)$ independent times. Hence by taking majority, we get the correct result, for each block, with failure probability that can be made to be smaller than $1/r^2$. Thus, in total of $O(m \cdot (n/m \cdot \log r)) = O(n \log r)$ broadcasts, all Z_i's are known to P_1 with error probability bounded by $1/r$.

In the next phase, P_1, which will manage the rest of the protocol, tries to find r 1-places. To do this he works in r rounds; at each he chooses a block B_i for which $Z_i = 1$. He then runs the OR protocol on it to find a next 1-place. He does this until he has collected r 1-places, or the OR on every block for which $Z_i = 1$ evaluates to 0. Formally, at the first round the protocol of Theorem 3 is repeated $\theta(\log r)$ times on the first block B_i for which $Z_i = 1$. Then as a result the majority value for the output index (if exists) is taken as a first 1-place. This totals in $O((n/m) \cdot \log r)$ broadcasts (to find the first 1-place). We now repeat the above, on the same block (excluding the 1-places that were already found), until the OR evaluates to 0 (or there is no majority index at the round above). At this point, we move to the next block until a total of r '1's are found, or all blocks result in 0 answer. Note that in order to carry out this protocol, P_1 needs faithfully distribute the next block to work on and the final results he receives. As there are r rounds, each results in an index $i \in [n]$, the faithful distribution cost is $O(r \log n)$ which is negligible in comparison with the actual computation: There are at most r such rounds, hence the total complexity of this phase is, $t = O(r \cdot (n/m) \log r) = O((n/r) \log r) = O(n)$.

Note that as described, the second phase of the protocol is not oblivious; at the ith round, it is not known in advanced what will be the block at hand. However, using t helpers, we construct an oblivious protocol using the same ideas as in the proof of Theorem 3. At the first round, every processor broadcasts its bit. From this point on, only the helpers will do the rest, in a similar way to that of Theorem 3. We leave further details on this issue for the final draft.

Finally, as each of the r rounds in the second phase is computed with error probability $1/r^2$, at the end all r 1-places are found correctly with probability at least $1 - 1/r$. Thus the overall error is bounded by $2/r < 1/3$ for large enough r. ■

6. Algorithms for OR of $\log^* n$ parallel time

We first describe here a fault-tolerant decision tree algorithm for the OR that has simultaneous $O(n)$ total complexity and $O(\log^* n)$ rounds. We then indicate how to simulate it in the broadcast model.

Theorem 8 *There is a fault-tolerant decision tree algorithm for the OR on n variables, of total complexity $O(n)$ and $O(\log^* n)$ rounds.*

proof: We assume along the sequel that $\epsilon < 1/4$. We adopt the strategy of the algorithm of Observation 2.2, but unfold the recursion using a somewhat different tradeoffs between number of queries and error probability. To present it concisely, we will have at each round $i = 0, 1, ...$ a set $I_i \subseteq [n]$ of variables that are supposed to hold the minimum index $j \in [n]$ for which $x_j = 1$ (if such j exists). Then at stage i every variable $x_j, j \in I_i$ is going to be queried k_i times, and a majority vote, M_j, on those values is going to represent its value for the ith round. Then based on those $M_j, j \in I_i$ the next subset of candidates $I_{i+1} \subseteq I_i$ is going to be formed as follows: we are going to divide I_i into blocks of size b_i each. Each block will contribute a single variable to I_{i+1} - namely the variable of minimum index in the block that has value '1' (or an arbitrary variable in the block if all evaluated to '0'). This is done until the subset of variable is narrowed down to a constant size. At this point by querying each variable $O(\log n)$ times and computing the OR on the majorities we are done. Note that as in the algorithm of Observation 2.2, this algorithm cannot err if the answer is '0'.

Formally, we denote by $n_i,\ i = 0, ...$ the size of I_i and set $k_0 = 100$ and $n_0 = n$. We denote by $b_i,\ i = 0, ..$ the size of the blocks at round i and let $b_i = exp(k_i/8) \cdot 2^{-(i+2)}$, $n_{i+1} = n_i/b_i$ and $k_{i+1} = 100n/n_{i+1} \cdot 2^{-(i+2)}$.

Thus assuming that the size of I_i is n_i, and that we divide I_i into blocks of size b_i from which one member is to enter I_{i+1} then the size of I_{i+1} is exactly n_{i+1}. To analyze the total complexity, recall that at round ℓ we query k_ℓ times each variable in I_ℓ. Thus, at round $i + 1$ a total of $n_{i+1} \cdot k_{i+1} = 100n2^{-(i+2)}$ queries are made. Thus the sum over all rounds is $O(n)$. To analyze the number of rounds, note that $\frac{k_{i+1}}{k_i} = 0.5\frac{n_i}{n_{i+1}} = 0.5b_i = 0.5 \cdot exp(k_i/8) \cdot 2^{-(i+2)}$ which means that k_i increases as $tower(i)$. In turn this means that b_i increases as a $tower$ too, which means that $n_{i+1} = n_i/b_i \leq n/b_i$ is bounded by constant for $i = O(\log^* n)$.

Finally, to analyze the error probability, assume that the OR is '1' and let us denote the minimum indexed variable whose value is '1' by A. Note that the algorithm is successful if for every round i, A is selected to enter the set I_i, and in addition, all variables at the end phase (each queried $\theta(\log n)$ times) evaluated correctly. The error probability of the last phase is certainly bounded by $o(1)$ (as there are only constant many variables left in I_j, for the last round j). Now, assuming that A is in I_{i-1}, the probability that it will not enter I_i is bounded by the probability that there is a variable in the block that contains A that gets evaluated erroneously. But as the individual error at round i for any variable evaluation is $exp(-k_i/8)$ (by Chernoff), and there are $b_i = exp(k_i/8) \cdot 2^{-(i+2)}$ variables in a block, the probability that there exists a variable in the block that contains A that gets wrongly evaluated is bounded by $2^{-(i+2)}$. Summing this for all rounds is still bounded by $1/4$. Hence the total error is bounded by $1/3$. ■

Theorem 9 *There is a $1/4$-fault tolerant protocol, in the fault tolerance broadcast model, that computes the OR on n variables in $O(\log^* n)$ rounds and total complexity $O(n)$.*

We don't give full details for the theorem. We only note that this is done using the same ideas as in the simulation done in Section 3 and observing that the faithful distribution of values can be parallelized, as only the parties within a block of round i need to know the values of the queries to variables in previous blocks that are contained in it.

7. Further work and open problems

We have demonstrated that via fault tolerance decision tree algorithms we can construct fault tolerance broadcasts protocols for many functions, including symmetric and non-symmetric examples. However, we still don't know the result to Gallager's main question in the general form. In particular we don't even know whether there is an $o(n \log\log n)$ algorithm to compute the all-bits problem for the general case. On the other hand, there are no trivial lower bounds.

One way to 'challenge' Gallager's upper bounds would be to generalize our results of recovering small weight words in complexity $o(n \log\log n)$. To do this for weights of $\sqrt{n}$ would be a major advance, while doing it for weights $\Omega(n/\log n)$ will solve the problem.

We showed that a good source of algorithms for the fault tolerance broadcast model is the fault tolerance decision tree model. This emphasize, to some extent, the importance of the decision tree parallel time as an important complexity measure. We note that the fault tolerance decision tree model is 'much weaker' then the broadcast; Feige et al. [5] proved a lower bound of $\Omega(n \log n)$ for the majority (and parity) function, while by Gallager those functions can be computed using $O(n \log\log n)$ broadcasts. However, as suitable algorithms transform from the decision tree model to the broadcast one, designing algorithms for the decision tree model is of importance. In this respect, linear (or $o(n \log\log n)$) algorithms are only known for either small minterms functions (or small maxterm), or to functions obtained from $O(1)$ levels of compositions of functions that are already known to be efficiently computable (see Observation 4.1). It would be nice to come up with different/other families for which efficient algorithms exist.

Finally, we nearly know everything about computing the OR function in both the noisy decision tree model and the broadcast model. There is still though one piece of information missing: Is it possible to compute the OR in $O(1)$ rounds and total complexity $O(n)$? This is currently not known even for the noisy decision tree model.

References

[1] R. L. Dobrushin and S. I. Ortyukov. Upper bounds for the redundancy of self-correcting arrangements of unreliable functions. *Problems of Information Transmission*, pages 203–218, 1977.

[2] A. El-Gamal. Open problems presented at the 1984 workshop on specific problems in communication and computation, sponsored by bell communication research. 1984.

[3] W. Evans and N. Pippenger. Lower bounds for noisy boolean decision trees. In *Proc. 28, STOC*, pages 620–628, 1996.

[4] U. Feige and J. Kilian. Finding or in noisy broadcast network. *IPL*, pages 69–75, 2000.

[5] U. Feige, P. Raghavan, D. Peleg, and E. Upfal. Computing with noisy information. *SIAM J. Comput.*, 23(5):1001–1018, 1994.

[6] P. Gács. Reliable computation with cellular automata. *JCSS*, 32:15–78, 1986.

[7] A. Gál. Lower bounds for the complexity of reliable boolean circuits with noisy gates. In *FOCS*, pages 594–601, 1991.

[8] R. G. Gallager. Finding parity in simple broadcast networks. *IEEE Trans. on Information Theory*, 34:176–180, 1988.

[9] S. Karlin and H. M. Taylor. *A First course in stochastic processes, Second Edition*. Academic Press, 1975.

[10] C. Kenyon and V. King. On boolean decision trees with faulty nodes. *Random Structures and Algorithms*, 5(3):453–464, 1994.

[11] E. Kushilevitz and Y. Mansour. Computation in noisy radio networks. In *SODA*, pages 236–243, 1998.

[12] A. V. Kuznetsov. Information storage in memory assembled form unreliable components. *Problems of Information Transmission*, 9:254–264, 1973.

[13] V. I. Levenšteĭn. Lower bounds for codes with a fixed weight of vectors (in russian). *Problemy Peredači Informacii*, 7:3–12, 1971.

[14] N. Pippenger. On networks of noisy gates. In *FOCS*, pages 30–38, 1985.

[15] R. Rajagopalan and L. J. Schulman. A coding theorem for distributed computation. In *STOC*, pages 790–799, 1994.

[16] R. Reischuk and B. Schmeltz. Reliable computation with noisy circuits and decision trees-a general n log n lower bound. In *FOCS*, pages 602–611, 1991.

[17] L. J. Schulman. Communication on noisy channels: A coding theorem for computation. In *FOCS*, pages 724–733, 1992.

[18] L. J. Schulman. Deterministic coding for interactive communication. In *STOC*, pages 747–756, 1993.

[19] D. A. Spielman. Highly fault tolerant parallel computation. In *FOCS*, pages 154–163, 1996.

[20] M. J. Taylor. Reliable information storage in memories designed form unreliable components. *Bell System Technical Journal*, 47:2299–2337, 1968.

[21] J. Von-Neumann. *Probabilistic logics and the synthesis of reliable organisms for unreliable components, Automated Studies, C. E. Shannon and J. McCarthy Eds*, pages 329–378. Princeton University Press, 1956.

[22] A. C. Yao. On the complexity of communication under noise. 1997.