Coding for Interactive Communication*

Leonard J. Schulman
Computer Science Division
U. C. Berkeley

Abstract

Let the input to a computation problem be split between two processors connected by a communication link; and let an interactive protocol π be known by which, on any input, the processors can solve the problem using no more than T transmissions of bits between them, provided the channel is noiseless in each direction. We study the following question: if in fact the channel is noisy, what is the effect upon the number of transmissions needed in order to solve the computation problem reliably?

Technologically this concern is motivated by the increasing importance of communication as a resource in computing, and by the tradeoff in communications equipment between bandwidth, reliability and expense.

We treat a model with random channel noise. We describe a deterministic method for simulating noiseless-channel protocols on noisy channels, with only a constant slow-down. This is an analog for general interactive protocols of Shannon's coding theorem, which deals only with data transmission, i.e. one-way protocols.

We cannot use Shannon's block coding method because the bits exchanged in the protocol are determined only one at a time, dynamically, in the course of the interaction. Instead we describe a simulation protocol using a new kind of code, explicit tree codes.

Key words: Interactive Communication, Coding Theorem, Tree Code, Distributed Computing, Reliable Communication, Error Correction.

*Special issue on Codes and Complexity of the IEEE Transactions on Information Theory, 42(6) Part I, 1745-1756, Nov. 1996.

1 Introduction

Let the input to a computation problem be split between two processors connected by a communication link; and let an interactive protocol π be known by which, on any input, the processors can solve the problem using no more than T transmissions of bits between them, provided the channel is noiseless in each direction. We study the following question: if in fact the channel is noisy, what is the effect upon the number of transmissions needed in order to solve the computation problem reliably?

We focus on a model in which the channel suffers random noise. In this case an upper bound on the number of transmissions necessary is provided by the simple protocol which repeats each transmission of the noiseless-channel protocol many times (and decodes each transmission by a vote). For this simulation to succeed, each transmission must be repeated enough times so that no simulated transmission goes wrong; hence the length of the simulation increases as a superlinear function of T, even if only a constant probability of error is desired. Said another way, the "rate" of the simulation (number of protocol steps simulated per transmission) goes to zero as T increases.

Shannon considered this matter in 1948 in his seminal study of communication [29]. Shannon studied the case of "one-way" communication problems, i.e. data transmission. His fundamental observation was that coding schemes which did not treat each bit separately, but jointly encoded large blocks of data into long codewords, could achieve very small error probability (exponentially small in T), while slowing down by only a constant factor relative to the T transmissions required by the noiseless-channel protocol (which can simply send the bits one by one). The constant (ratio of noiseless to noisy communication time) is a property of the channel, and is known as the Shannon capacity of the channel. The improvement in communication rate provided by Shannon's insight is dramatic: the naive protocol can only achieve the same error probability by repeating each bit a number of times proportional to the length of the entire original protocol (for a total of $\Omega(T^2)$ communications[1]).

A precise statement is captured in Shannon's coding theorem:

Theorem 1 (Shannon) *Let a binary symmetric channel of capacity C be given. For every T and every $\gamma > 0$ there exists a code $\chi : \{0,1\}^T \to \{0,1\}^{T\frac{1}{C}(1+\gamma)}$ and a decoding map $\chi' : \{0,1\}^{T\frac{1}{C}(1+\gamma)} \to \{0,1\}^T$ such that every codeword transmitted across the channel is decoded correctly with probability $1 - e^{-\Omega(T)}$.*

The original proof of such a theorem appears in [29] (formulated already for the more general class of Markov channels). The ramifications of this insight for data transmission have been explored in coding theory and information theory.

Recently, in computer science, communication has come to be critical to distributed computing, parallel computing, and the performance of VLSI chips. In these contexts interaction is an essential part of the communication process. If the environment is noisy, it is necessary to be able to sustain interactive communication in the presence of that noise. Noise afflicts interactive communications just as it does the one-way communications considered by Shannon, and for much the same reasons: physical devices are by nature noisy, and there is often a significant cost associated with making them so reliable that the noise can be ignored (such as by providing very strong transmitters, or using circuits cooled to very low temperatures). In order to mitigate such costs we must design our systems to operate reliably even in the presence of some noise. The ability to transmit data in the presence of noise, the subject of Shannon's and subsequent work, is a necessary but far from sufficient condition for sustained interaction and computation.

For this reason we will be concerned with the problem of achieving simultaneously high communication rate and high reliability, in arbitrary interactive protocols.

[1] The following notation will be used in this paper. If $g : \mathbb{N} \to \mathbb{N}$ is a nondecreasing function, then $\Omega(g)$ denotes the set of functions $f : \mathbb{N} \to \mathbb{N}$ such that $\liminf f(i)/g(i) > 0$; $O(g)$ denotes the set of functions $f : \mathbb{N} \to \mathbb{N}$ such that $\limsup f(i)/g(i) < \infty$; and $\theta(g) = O(g) \cap \Omega(g)$, the set of functions which, asymptotically, grow proportionately to g.

Observe that in the case of an interactive protocol, the processors generally do not know what they want to transmit more than one bit ahead, and therefore cannot use a block code as in the one-way case. Another difficulty that arises in our situation, but not in data transmission, is that once an error has occurred, subsequent exchanges on the channel are affected. Such exchanges cannot be counted on to be of any use either to the simulation of the original protocol, or to the detection of the error condition. Yet the processors must be able to recover, and resume synchronized execution of the intended protocol, following any sequence of errors, although these may cause them to have very different views of the history of their interaction.

We describe in this paper how, in spite of these new difficulties, it is possible to simulate noiseless-channel protocols on noisy channels with exponentially small error and with an increase by only a constant factor in the number of transmissions. First we state a version of our result for stochastic noise. In this theorem, as elsewhere in the paper except section 5, we assume that in both the noisy and noiseless scenarios, the processors are connected by a pair of unidirectional channels which, in every unit of time, transmit one bit in each direction. The run time of a protocol is the number of bit exchanges required for it to terminate on any input.

Theorem 2 *In each direction between a pair of processors let a binary symmetric channel of capacity C be given. There is a deterministic communication protocol which, given any noiseless channel protocol π running in time T, simulates π on the noisy channel in time $\theta(T/C)$ and with error probability $e^{-\Omega(T)}$.*

Note that if π is deterministic, so is the simulation; if π takes advantage of some randomness then exactly the same randomness resources are used by the simulation, and the probability of error will be bounded by the sum of the probability of error of π and the probability of error of the simulation.

In all but a constant factor in the rate, this is an exact analog, for the general case of interactive communication problems, of the Shannon coding theorem. We focus in this paper on binary channels as the canonical case; but we do not restrict ourselves to memoryless channels or even to stochastic noise, as will be seen below.

We will also show that if the "tree codes" introduced in this paper can be efficiently constructed, then the encoding and decoding procedures of the protocol can also be implemented efficiently:

Theorem 3 *Given an oracle for a tree code, the expected computation time of each of the processors implementing our protocol, when the communication channels are binary symmetric, is polynomial in T.*

Tree codes will be described below. Following standard usage in complexity theory, an "oracle" is any procedure which, on request, produces a local description of the tree code; one unit of time is charged for this service. Thus any polynomial time algorithm for this task will result in polynomial time encoding and decoding procedures.

It is worth noting that if all one wants is to transmit one bit (or a fixed number of bits) but a very small probability of error is desired, then there is only one thing to be done: the redundancy of the transmission must be increased, in order to drive down the error probability. (By redundancy we mean the number of transmissions per bit of the message.) Shannon's theorem says that this is not necessary if one may assume that the message to be transmitted is long. In that case a code can be selected in such a manner that when any codeword (an image of a T-bit input block via χ, as above) is transmitted over the channel, the probability that it is changed by the channel noise to a string which is mistaken by the receiver for another codeword, is exponentially small in T. Since the exponential error is gained from the block size, T, which may be chosen as large as desired, it is not necessary to invest extra redundancy for this purpose. It is enough just to bring the redundancy above the threshold value $1/C$.

For a noiseless environment, the interactive model for communication problems, in which the input to a problem is split between two processors linked by a noiseless channel, was introduced by A. Yao in

1979 [36] to measure the cost incurred in departing from the single-processor model of computation. It has been intensively investigated (e.g. [37, 19, 32, 17]; and see [18] for a survey). The present work can be viewed as guaranteeing that every upper bound in that model, yields an upper bound in the noisy model.

A weaker interactive analog of Shannon's coding theorem, which made use of randomization in the simulation process, was given by the author in [26] and [27]. The present result was presented in preliminary form in [28].

The binary symmetric channel (BSC) is a simple noise model which nevertheless captures the chief difficulties of the problem considered in this paper; therefore we have focussed on it. However with little additional effort our result can be extended to far more general channels.

Theorem 4 (Adversarial Channel) *There is a deterministic communication protocol which, given any noiseless channel protocol π running in time T, runs in time $N = \theta(T)$, and successfully simulates π provided the number of incorrectly transmitted bits is at most $N/240$.*

(Note that due to the limit on the tolerable noise level this result does not dominate theorem 2.)

We also consider in this paper what happens in a noise model that is much milder than the BSC: binary erasure channels with free feedback. In this case a particularly clean result can be obtained, which will be described in section 5.

The model

In the standard (noiseless) communication complexity model, the argument (input) $z = (z_A, z_B)$ of a function $f(z)$ is split between two processors A and B, with A receiving z_A and B receiving z_B; the processors compute $f(z)$ (solve the communication problem f) by exchanging bits over a noiseless link between them. Each processor has unbounded time and space resources. Randomness may be allowed as a resource, and is typically considered at one of three levels: none (deterministic protocols), private coins (each processor has its own unbounded source of random coins, but cannot see the other processor's coins), and public coins (the processors can see a common unbounded source of random coins). The procedure which determines the processors' actions, based on their local input and past receptions, is called a communication protocol. We will be concerned in this paper with a deterministic simulation of one protocol by another. In case the original protocol was deterministic, so is the simulation. If a randomized protocol is to be simulated, the same method applies; simply choose the random string in advance, and then treat the protocol being simulated as if it were deterministic.

In this paper the noiseless link is replaced (in each direction) by a noisy one. We will mostly focus on the following noise model. The *binary symmetric channel* is a synchronous communication link which, in every unit of time, accepts at one end either a 0 or 1; and in response produces either a 0 or a 1 at the other end. With some fixed probability ϵ the output differs from the input, while with probability $1-\epsilon$ they are the same. This random event for a particular transmission is statistically independent of the bits sent and received in all other channel transmissions. (Thus the channel is "memoryless".) Weaker and stronger assumptions regarding the channel noise will be considered in sections 3.1 and 5.

The average mutual information between two ensembles $\{P(x)\}_{x\in X}$ and $\{P(y)\}_{y\in Y}$, which have a joint distribution $\{P(xy)\}_{x\in X, y\in Y}$, is defined as $\sum_{xy} P(xy) \log \frac{P(xy)}{P(x)P(y)}$. This is a measure of how much information is provided about one ensemble by the specification of a point (chosen at random from the joint distribution) of the other ensemble.

The capacity of a channel was defined by Shannon as the maximum, ranging over probability distributions on the inputs, of the average mutual information between the input and output distributions. In the case of the binary symmetric channel the capacity (in base 2) is $C = \epsilon \lg(2\epsilon) + (1-\epsilon)\lg(2(1-\epsilon))$. Observe that $0 \le C \le 1$; and that a noiseless channel has capacity $C = 1$.

2 Coding Theorem

A key tool in our protocol is the tree code. Random tree codes — essentially, distributions over labelled trees — were introduced by Wozencraft and used by him and others for the purpose of sequential decoding ([34, 24, 7]; see [12] §6.9). However, random tree codes are not sufficient for our purpose. We introduce below the stronger notion of tree code which we use in our work.

2.1 Tree codes

Let S be a finite alphabet. If $s = (s_1...s_m)$ and $r = (r_1...r_m)$ are words of the same length over S, say that the distance $\Delta(s,r)$ between s and r is the number of positions i in which $s_i \neq r_i$ (Hamming distance). A d-ary tree of depth n is a rooted tree in which every internal node has d children, and every leaf is at depth n (the root is at depth 0).

Definition *A d-ary tree code over alphabet S, of distance parameter α and depth n, is a d-ary tree of depth n in which every arc of the tree is labeled with a character from the alphabet S subject to the following condition. Let v_1 and v_2 be any two nodes at some common depth h in the tree. Let $h - \ell$ be the depth of their least common ancestor. Let $W(v_1)$ and $W(v_2)$ be the concatenation of the letters on the arcs leading from the root to v_1 and v_2 respectively. Then $\Delta(W(v_1), W(v_2)) \geq \alpha\ell$.*

In the codes we will use in this paper we fix the distance parameter α to $1/2$.

The key point regarding tree codes is that the alphabet size required to guarantee their existence does not depend on n. In fact we will show that for any fixed d there exists an infinite tree code with a constant size alphabet (i.e. a complete, infinite d-ary tree in which the labels satisfy the distance condition).

We begin with the case of $d = 2$ and finite n; then treat arbitrary d and an infinite tree.

Lemma 1 *For any fixed degree d, there exists a finite alphabet which suffices to label the arcs of a d-ary tree code. Specifically:*

(A) An alphabet S of size $|S| = 99$ suffices to label the arcs of a binary tree code of distance parameter $1/2$ and any depth n.

Let $\eta(\alpha) = \frac{1}{(1-\alpha)^{1-\alpha}} \frac{1}{\alpha^\alpha}$. Note that $\eta(\alpha) \leq 2$.

(B) An alphabet S of size $|S| = 2\lfloor (2\eta(\alpha)d)^{\frac{1}{1-\alpha}} \rfloor - 1$ suffices to label the arcs of an infinite d-ary tree code of distance parameter α.

Proof:

(A) By induction on n. A base case $n = 0$ for the induction is trivial. Now take a tree code of depth $n-1$, and put two copies of it side-by-side in a tree of depth n, each rooted at a child of the root. Choose a random string $\sigma = (\sigma_1...\sigma_n)$, where the variables $\{\sigma_h\}$ are i.i.d. random variables, each chosen uniformly from among the set of permutations of the letters of the alphabet. Modify the right-hand subtree by replacing each letter x at depth h in the subtree, by the letter $\sigma_h(x)$. (Also place the letters 1 and $\sigma_1(1)$ on the arcs incident to the root.) This operation does not affect the tree-code conditions within each subtree. Now consider any pair of nodes v_1, v_2 at depth h, with v_1 in the left subtree and v_2 in the right subtree. The probability that they violate the tree-code condition is bounded by $\exp(-hD(1-\alpha||1/|S|))$, where D is the Kullback-Leibler divergence, $D(x||y) = x\log x/y + (1-x)\log(1-x)/(1-y)$. The number of such pairs at depth h is 4^{h-1}. Thus the probability of a violation of the tree code condition at any depth is bounded by $\sum_{h=1}^{\infty} \exp((h-1)2\log 2 - h(D(1-\alpha||1/|S|)))$. For the case $\alpha = 1/2$ and $|S| = 99$ this is is strictly less than 1 (approximately .99975). Hence some string σ produces a tree code of depth n.

(B) What follows is the best currently known existence argument for tree codes[2]. Let $|S|$ be the smallest prime greater or equal to $(2\eta(\alpha)d)^{\frac{1}{1-\alpha}}$. By Bertrand's postulate [14], there is a prime strictly between n and $2n$ for any integer $n > 1$. Hence $|S| < 2\lfloor(2\eta(\alpha)d)^{\frac{1}{1-\alpha}}\rfloor$.

Identify S with the set $\{0, 1, ..., |S|-1\}$. Associate with every arc of the tree its address, which for an arc at depth h, is the string in $\{0, ..., d-1\}^h$ describing the path from the root to that arc. (Thus the arcs below the root are labelled $\{0, ..., d-1\}$; the arcs below the 0-child of the root are labelled $\{00, ..., 0(d-1)\}$; and so forth.)

Let $a = a_1a_2a_3...$ be an infinite string with each a_i chosen uniformly, independently, in $\{0, ..., |S|-1\}$. Label the tree code by convolving the arc addresses with the random string, as follows: if the address of an arc (at depth h) is $\varepsilon = \varepsilon_1\varepsilon_2...\varepsilon_h$, then the arc is labelled $\sum_{i=1}^{h} \varepsilon_i a_{h+1-i}$.

Let $i, j \in \{0, ..., d-1\}$, $i \neq j$; and let x, y, z be strings over $\{0, ..., d-1\}$, with $|y| = |z| = r$. Observe that

$$\text{label}(xiy) - \text{label}(xjz) = (i-j)a_{r+1} + \sum_{i=1}^{r}(y_i - z_i)a_{r+1-i}.$$

Therefore the event that xiy and xjz fail the distance condition depends only on the sequence $(i-j), (y_1 - z_1), ..., (y_r - z_r)$. The probability that a causes a failure of the distance condition on this sequence is bounded by $\exp(-hD((1-\alpha)||(1/|S|)))$ (where $h = r+1$); summing over all such sequences, we bound the probability that a fails to provide a tree code by $\sum_{h\geq 1} d^h \exp(-hD((1-\alpha)||(1/|S|)))$. This in turn is strictly less than $\sum_{h\geq 1} \exp(h(\log d - (1-\alpha)\log|S| - (1-\alpha)\log(1-\alpha) - \alpha\log\alpha))$ which, with the stated choice of $|S|$, is bounded by 1. Therefore there is a positive probability that the string a defines an infinite tree code. □

What is needed for efficient implementation of our protocol, is a tree code in which the label on any edge can be computed in time polynomial in the depth of the tree, or (for an infinite tree) in the depth of the edge.

An "oracle", as referred to in theorem 3, is an abstract machine which provides, on request, labels of edges in the tree code; we are charged one unit of computation time per request. Therefore theorem 3 guarantees that, should we be able to come up with a polynomial computation-time implementation of a tree code, this would result in a polynomial computation-time coding theorem for noisy channels.

2.2 Preliminaries

2.2.1 The role of channel capacity

A standard calculation shows:

Lemma 2 (standard) *Let a binary symmetric channel M of capacity C, and an alphabet S, be given. Then there is a transmission code for the alphabet, i.e. a pair consisting of an encoding function $\kappa : S \to \{0,1\}^n$ and a decoding function $\kappa' : \{0,1\}^n \to S$, such that $n = O(\frac{1}{C}\log|S|)$, while the probability of error in any transmission over the channel (i.e. the probability that $\kappa'(M(\kappa(s))) \neq s$) is no more than $2^{-208}3^{-40}$.* □

In the protocol we make use of a 12-ary tree code with distance parameter 1/2. The protocol as we describe it will involve the processors transmitting, in each round, a letter inscribed on some arc of the

[2]The present form of this argument is due to discussions with Yuval Rabani (for the convolution) and Oded Goldreich (for the $1/(1-\alpha)$ exponent).

tree; this will be implemented by the transmission of the codeword for that letter, in a transmission code as described above. (Thus, due to lemma 1, we will be applying lemma 2 with a constant size alphabet.) This presentation slightly simplifies our later description of the protocol, and also makes clear how the channel capacity enters the slow-down factor of theorem 2.

The role of the channel capacity in our result should not be overemphasized. In contrast to the situation for one-way communication, where the capacity provides a tight characterization of the rate at which communication can be maintained, in the interactive case we will be able to characterize this rate only to within a constant factor of capacity. Moreover the positive results of this paper use only crude properties of the capacity: its quadratic basin (as a function of the channel error probability) in the very noisy regime, and its boundedness elsewhere. The interesting problem remains, whether there are protocols whose simulation in the presence of noise must slow down by a factor worse than the capacity.

2.2.2 Noiseless Protocol π

Let π be the noiseless-channel protocol of length T to be simulated. We assume the "hardest" case, in which every transmission of π is only one bit long; and for simplicity we suppose that in every round both processors send a bit (this affects the number of transmissions by no more than a factor of 2).

The history of π on any particular input is described by a path from the root to a leaf, in a 4-ary tree (which we denote $\mathcal{T}$, see figure 1) in which each arc is labelled by one of the pairs $00, 01, 10$ or 11 — referring to the pair of messages that can be exchanged by the processors in that round. (Thus, e.g., 10 refers to the transmission of a 1 by processor A and a 0 by processor B.) If $x = x_A x_B$ is the problem input (x_A at processor A and x_B at processor B) then let $\pi_A(x_A, \phi)$ denote the first bit sent by processor A; let $\pi_B(x_B, \phi)$ denote the first bit sent by processor B; and let $\pi(x, \phi) \in \{00, 01, 10, 11\}$ denote the pair of these first-round bits. In the second round of π both processors are aware of the bits exchanged in the first round, and so we denote their transmissions by $\pi_A(x_A, \pi(x, \phi))$ and $\pi_B(x_B, \pi(x, \phi))$ respectively, and the pair of bits by $\pi(x, \pi(x, \phi))$. In general if messages $m_1, ..., m_t$ (each $m_i \in \{00, 01, 10, 11\}$) have been exchanged in the first t rounds, then we denote their transmissions in round $t+1$ by $\pi_A(x_A, m_1 ... m_t)$ and $\pi_B(x_B, m_1 ... m_t)$, and the pair of bits by $\pi(x, m_1 ... m_t)$. The vertex of $\mathcal{T}$ labelled by a sequence of exchanges $m_1 ... m_t$ is denoted $\mathcal{T}[m_1 : ... : m_t]$. For example the root is $\mathcal{T}[\,]$.

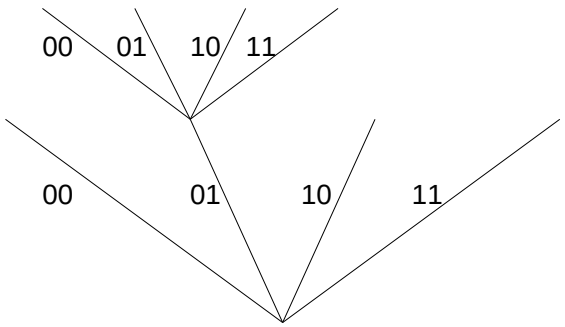

Figure 1: Noiseless protocol tree $\mathcal{T}$.

On input x the protocol π specifies a path from the root to a leaf in $\mathcal{T}$, namely the succession $\mathcal{T}[\,]$, $\mathcal{T}[\pi(x, \phi)]$, $\mathcal{T}[\pi(x, \phi) : \pi(x, \pi(x, \phi))]$, etc., at the end of which the outcome of the protocol is determined. We call this path γ_x. On a noiseless channel the processors simply extend γ_x by one arc in every round. In the noisy channel simulation the processors must develop γ_x without expending too much time pursuing other branches of $\mathcal{T}$.

2.3 Simulation Protocol

The simulation protocol attempts, on input $x = x_A x_B$, to recreate the development by π of the path γ_x in $\mathcal{T}$. We equip each processor with a pebble which it moves about on $\mathcal{T}$, in accordance with its "best guess" to date regarding γ_x. In every round, each processor sends the bit it would transmit in π upon reaching the current pebble position in $\mathcal{T}$ (among other information). When the pebbles coincide, this exchange is a useful simulation of a step in π (unless the exchange fails). However, due to channel errors, the processors' pebbles will sometimes diverge. The simulation embeds π within a larger protocol which provides a "restoring force" that tends to drive the pebbles together when they separate. The guarantee that (with high probability) the pebbles coincide through most of the simulation is what ensures the progress of the simulation.

We begin by describing the mechanics of the protocol: how the pebbles can be moved about, what information the processors send each other as they move the pebbles, and how they use a tree code to do so. Then we explain how the processors choose their pebble moves based upon their local input and their history of the interaction.

One way to think about our protocol is that it tries to make the sequence of transmissions produced by each processor, mimic the sequence that might be produced by a source transmitting a large block of data (and thus subject to efficient coding), in spite of the fact that the protocol is interactive and processors do not know the future "data" they will be transmitting.

2.3.1 The state tree: mechanics of the protocol

In any round of the protocol, a processor can move its pebble only to a neighbor of the current position in $\mathcal{T}$, or leave it put. Thus any move is described by one of the six possibilities $00, 01, 10, 11, \mathcal{H}$ ("hold") and $\mathcal{B}$ ("back", i.e. toward the root.) Following such a move, as mentioned above, the simulation involves transmitting either a 0 or a 1 according to the value transmitted in π at that point. (E.g., upon reaching $v \in \mathcal{T}$, A needs to inform B of $\pi_A(x_A, v)$.)

The entire history of any one processor up through time t can therefore be described as a sequence of t "tracks", each an integer between 1 and 12, indicating the pebble move made by the processor in some round, and the value $\pi_A(x_A, v)$ (or correspondingly $\pi_B(x_B, v)$) at the vertex v reached by the pebble after that move. In particular, the position of the processor's pebble can be inferred from this sequence of tracks. Let us say that the sequence of tracks taken by a processor is its "state": thus the universe of possible states for a processor is described by a 12-ary tree which we call $\mathcal{Y}$ (see figure 2), and in each round, each processor moves to a child of its previous state in this "state tree". (It will suffice to take $\mathcal{Y}$ of depth $5T$.) For a state $s \in \mathcal{Y}$ let pebble$(s) \in \mathcal{T}$ denote the pebble position of a processor which has reached state s.

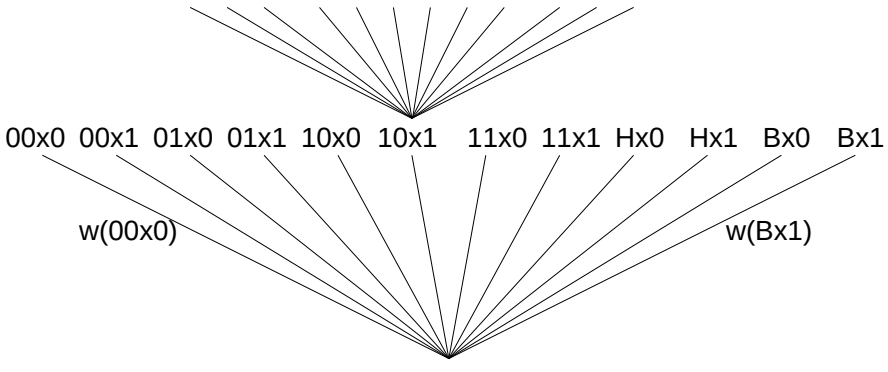

Figure 2: State tree $\mathcal{Y}$ and associated tree code

The processors' strategy will be founded on a "total information" approach: they will try to keep each other informed of their exact state, and thus of their entire history. Naturally, they cannot do this simply by encoding each track with a transmission code and sending it across the channel; in order to achieve the desired constant slow-down, such a code could have only constant length, hence channel errors would cause occasional errors in these track transmissions in the course of the simulation. (These would in fact occur at a constant frequency.) In the absence of any further indication of the correctness or incorrectness of the decoding of any track transmission, each processor would be entirely incapable of recovering the other's sequence of tracks.

Instead it is necessary to transmit the tracks using constant length encodings which, in spite of their brevity, incorporate global information about the track sequence, and thus enable recovery from errors in individual track transmissions. We accomplish this with the help of tree codes. Following lemma 1 identify with $\mathcal{Y}$ a 12-ary tree code of depth $5T$ over suitable alphabet S. If $\tau_1...\tau_t$ is a sequence of t tracks (indicating a processor's actions in the first t rounds) then $\mathcal{Y}[\tau_1:...:\tau_t]$ denotes the vertex of $\mathcal{Y}$ reached from the root by the sequence of arcs $\tau_1...\tau_t$. Denote by $w(\tau_1...\tau_t) \in S$ the letter on arc $\mathcal{Y}[\tau_1:...:\tau_{t-1}]\mathcal{Y}[\tau_1:...:\tau_t]$ of the tree code, and by $W(\tau_1...\tau_t) \in S^t$ the concatenation $w(\tau_1)w(\tau_1\tau_2)...w(\tau_1...\tau_t)$.

The prescription for transmitting messages in the protocol is now this: suppose processor A in round t decides on a pebble move $\alpha \in \{00, 01, 10, 11, \mathcal{H}, \mathcal{B}\}$, and reaches vertex v of $\mathcal{T}$ with that move; thus its track is $\tau_t = \alpha \times \pi_A(x_A, v)$, corresponding to an integer between 1 and 12. Further suppose its past tracks were $\tau_1, ..., \tau_{t-1}$. Then it transmits in round t the letter $w(\tau_1...\tau_t)$.

If $\tau_1...\tau_t$ is a sequence of tracks (a state), and $Z = Z_1...Z_t \in S^t$ is a sequence of letters, then let $P(Z|W(\tau_1...\tau_t))$ denote the probability that Z is received over the channel given that the sequence $W(\tau_1...\tau_t)$ was transmitted. More generally for $r \geq 1$ let $P(Z_r...Z_t|W(\tau_1...\tau_t))$ denote the probability that $Z_r...Z_t$ are the last $t-r+1$ characters received in a transmission of $W(\tau_1...\tau_t)$. Observe that these probabilities are the product of terms each equal to the probability that a particular letter is received, given that some other was transmitted; each term depends only on the channel characteristics and the transmission code (given by κ, κ'). Due to the memorylessness of the channel and lemma 2 we have:

Lemma 3 *$P(Z|W(\tau_1...\tau_t))$ is bounded above by $(2^{-208}3^{-40})^{\Delta(W(\tau_1...\tau_t),Z)}$. More generally $P(Z_r...Z_t|W(\tau_1...\tau_t)) \leq (2^{-208}3^{-40})^{\Delta(w(\tau_1...\tau_r)...w(\tau_1...\tau_t),Z_r...Z_t)}$.* □

2.3.2 Pebble Moves

We now specify how the processors choose their pebble moves. Our description is from the perspective of processor A, everything for B is symmetric. Let $Z \in S^{t-1}$ be the sequence of letters received by A up to and including round $t-1$. A then determines which state $\tau_1...\tau_{t-1}$ of processor B at depth $t-1$, minimizes the distance $\Delta(W(\tau_1...\tau_{t-1}), Z)$, and chooses that state g as its best guess for B's current state. Observe that a correct guess implies correct determination of B's pebble position.

(We have used here a minimum-distance condition; max-likelihood would work, as well.)

Suppose A's pebble is at vertex $v \in \mathcal{T}$, and that following the most recent exchange it has guessed that B's state is $g \in \mathcal{Y}$. Recall that A has just sent the bit $\pi_A(x_A, v)$ to B; and let b be the corresponding bit which A has decoded as B's proposed message in π. Processor A now compares the positions of v and pebble(g) in $\mathcal{T}$, and, acting on the presumption that its guess g is correct, chooses its next pebble move as follows.

If v and pebble(g) are different then A tries to close the gap by making the pebbles meet at their least common ancestor in $\mathcal{T}$. There are two cases. If v is not an ancestor of pebble(g) then A moves its pebble toward the root in $\mathcal{T}$ ("back"). If v is an ancestor of pebble(g) then A cannot close the gap itself, and instead keeps its pebble put in the expectation that B will bring its pebble back to v.

If on the other hand $v = \text{pebble}(g)$ then A interprets b as the bit B would transmit at v in protocol π. Therefore it moves its pebble down in $\mathcal{T}$ on the arc labelled by the pair of bits given by b, and its own choice $\pi_A(x_A, v)$.

The focus upon the least common ancestor of the pebbles is explained by the following.

Lemma 4 *The least common ancestor of the two pebbles lies on γ_x.*

Proof: A vertex z of $\mathcal{T}$ is on γ_x if and only if for every strict ancestor z' of z, the arc leading from z' toward z is that specified by both processors' actions in π at z', namely the arc labeled by the pair of bits $\pi_A(x_A, z')$ and $\pi_B(x_B, z')$. The pebble moves specified above ensure that the arcs leading toward a pebble are always correctly labelled in the bit corresponding to that processor's actions in π. □

We collect the above in a concise description of the protocol, from the perspective of processor A.

2.3.3 Summary: Simulation Protocol

Repeat the following $N = 5T$ times (where T is the length of protocol π). Begin with own state s_A at $\mathcal{Y}[\mathcal{H} \times \pi_A(x_A, \phi)]$, and own pebble at the root of $\mathcal{T}$.

1. Transmit $w(s_A)$ to processor B.
2. Given the sequence of messages Z received to date from processor B, guess the current state g of B as that minimizing $\Delta(W(g), Z)$. Compute from g both pebble(g), and the bit b (representing a message of B in π).
3. Depending on the relation of v to pebble(g), do one of the following:
   (a) If v is a strict ancestor of pebble(g), own pebble move is $\mathcal{H}$. Own next track is $\tau = \mathcal{H} \times \pi_A(x_A, v)$. Reset s_A to $s_A\tau$.
   (b) If the least common ancestor of v and pebble(g) in the state tree is a strict ancestor of v, then own pebble move is $\mathcal{B}$. Own next track is $\tau = \mathcal{B} \times \pi_A(x_A, v{:}\mathcal{B})$. Reset s_A to $s_A\tau$. (Here $v{:}\mathcal{B}$ denotes the parent of v in $\mathcal{T}$, i.e. the vertex reached after a "back" move.)
   (c) If $v = \text{pebble}(g)$ then move own pebble according to the pair of bits $(\pi_A(x_A, v), b)$. Own next track is $\tau = (\pi_A(x_A, v), b) \times \pi_A(x_A, v{:}(\pi_A(x_A, v), b))$. Reset s_A to $s_A\tau$. (Here $v{:}(\pi_A(x_A, v), b)$ denotes the child of v in $\mathcal{T}$ reached by the move $(\pi_A(x_A, v), b)$.)

A technical point: for the purpose of the simulation we modify π so that, once the computation is completed, each processor continues sending 0's until time $5T$. Thus we consider $\mathcal{T}$ as having depth $5T$, with the protocol tree for the unmodified π embedded within the first T levels. This ensures that the above process is meaningful for any state of processor A. In view of lemma 4, the simulation is successful if both pebbles terminate at descendents of the same "embedded leaf".

3 Analysis

We prove the following bound on the performance of the protocol, for binary symmetric channels. Theorem 2 follows in consideration of the comments in section 2.2.1 on the coding of individual rounds.

Proposition 1 *The simulation protocol, when run for $5T$ rounds on any noiseless-channel protocol π of length T, will correctly simulate π except for an error probability no greater than 2^{-5T}.*

Proof: Let v_A and v_B denote the positions of the two pebbles in $\mathcal{T}$ after some round of the simulation. The least common ancestor of v_A and v_B is written $\bar{v}$.

Definition *The current* mark *of the protocol is defined as the depth of $\bar{v}$, minus the distance from $\bar{v}$ to the further of v_A and v_B.*

Observe that π is successfully simulated if the mark at termination is at least T.

Definition *Say that a round is* good *if both processors guess the other's state correctly. Otherwise say that the round is* bad.

These definitions are related through the following:

Lemma 5

1. *The pebble moves in a good round increase the mark by 1.*
2. *The pebble moves in a bad round decrease the mark by at most 3.*

Proof:

1. After a good round the processors are using the correct information regarding both the other processor's pebble position, and the bit the other processor transmits in π at that position.

   If the pebbles were not located at the same vertex, then either pebble not equal to $\bar{v}$ moves one arc closer to it. If the pebbles were located at the same vertex then each progresses to the same child of that vertex.

2. Each pebble (and hence also $\bar{v}$) moves by at most one arc. □

Recall that the simulation is run for $N = 5T$ rounds.

Corollary 1 *The simulation is successful provided the fraction of good rounds is at least* $4/5$. □

The task of bounding the number of bad rounds is complicated by the fact that this quality is not independent across rounds.

For any round t let $\ell(t)$ be the greater, among the two processors, of the magnitude of their error regarding the other's state; this magnitude is measured as the difference between t, and the level of the least common ancestor of the true state and the guessed state. Thus a round is good precisely if $\ell(t) = 0$. Due to the tree code condition, any wrong guess of magnitude ℓ is at Hamming distance at least $\ell/2$ from the correct guess; in order to be preferred to the correct guess, at least $\ell/4$ character decoding errors are required within the last ℓ rounds. There are at most 2^ℓ ways in which these errors can be distributed. Applying lemma 3 — and noting that this application relies only on the transmissions and receptions during the ℓ rounds in question, and that the channel is memoryless — we have:

Corollary 2 *Conditioned on any pair of states the processors may be in at time $t - \ell$, the probability of a processor making a particular wrong guess of magnitude ℓ at time t is at most* $2^\ell (2^{-208} 3^{-40})^{\ell/4} = (2^{-51} 3^{-10})^\ell$. □

Now define the *error interval* corresponding to a bad round at time t as the sequence of $\ell(t)$ rounds $t - \ell(t) + 1, ..., t$. Every bad round is contained inside an error interval, hence the size of the union of the error intervals is a bound on the number of bad rounds. The remainder of the proof rests on the following pair of observations:

Lemma 6 *If a set of erroneous guesses define a disjoint set of error intervals, of lengths $\ell_1, ..., \ell_k$, then the probability that these erroneous guesses occur is at most $(2^{-50}3^{-10})^{\Sigma \ell_i}$.*

Proof: The argument is by induction on the number of error intervals. Let the last error interval be due to a wrong guess at round t, of magnitude ℓ_k. Let Z be the received sequence through round t. Let s be the true state of the other processor at time t, and let r be the erroneous guess. Then the transmitted strings $W(s)$ and $W(r)$ differ only in the last ℓ_k rounds. Corollary 2 implies that the bound on the probability of occurrence of the erroneous guesses is multiplied by a factor of $2(2^{-51}3^{-10})^{\ell_k}$ (the factor of 2 allows for either processor to be in error). □

Lemma 7 *In any finite set of intervals on the real line whose union J is of total length s there is a subset of disjoint intervals whose union is of total length at least $s/2$.*

Proof: We show that J can be written as the union of two sequences of disjoint intervals.

The question reduces to the case in which the intervals of the family are closed and their union J is an interval. In the first step put into the first sequence that interval which reaches the left endpoint of J, and which extends furthest to the right. In each successive step select the interval which intersects the union of those selected so far, and which extends furthest to the right; adjoin the new interval to one of the sequences in alternation. □

Lemma 7 and the sufficiency condition of corollary 1 reduce our problem to the task of bounding the probability of occurrence of some set of erroneous guesses which define disjoint error intervals spanning at least $N/10$ rounds. First consider all ways in which these erroneous guesses can arise. There are at most 2^{2N} ways in which the disjoint error intervals can be distributed in the N rounds available. For each error of magnitude ℓ, the erroneous guess can be at one of $12^\ell - 1$ states; hence for a given set of disjoint error intervals there are, in all, at most 12^N ways in which the guesses can arise.

If a set of erroneous guesses defines a set of disjoint error intevals of lengths $\ell_1, ...\ell_k$ then by corollary 2 and lemma 6 the probability of all these guesses occurring is at most $96^{-10\sum \ell_i}$. In particular if these error intervals span at least $N/10$ rounds then the probability of this event is at most 96^{-N}. Ranging over all such events we find that the probability of the protocol failing to simulate π is at most $2^{2N}12^N96^{-N} = 2^{-N} = 2^{-5T}$. □

3.1 Adversarial Channel

Proof of Theorem 4: For the transmission code used to exchange individual tracks between the processors (thus individual letters of S, the tree code alphabet), we use a code with a minimum-distance property, so that any two codewords differ in at least 1/3 of their length. (Thus the protocol here differs slightly from that for the BSC, where we use a code that achieves small error probability, not necessarily a minimum-distance code.)

In order that one of the processors make a wrong guess of magnitude ℓ, at least $\ell/4$ of the transmitted characters must have been received incorrectly, since we are using a minimum-distance condition and the number of differing characters between the correct and incorrect branches is at least $\ell/2$. Since for the character transmissions we are using the above minimum-distance code, it follows that the fraction of incorrectly transmitted bits during this period of length ℓ, must be at least 1/24.

To avoid double-counting these errors we again resort to lemma 7, and wish to bound the total length of disjoint error intervals by $N/10$. For this it therefore suffices that the fraction of bit errors during the protocol be bounded by 1/240. □

4 Computational Overhead of the Protocol

Proof of Theorem 3: The computationally critical aspect of the simulation protocol is the determination, in each round, of the branch of the tree which minimizes the distance to the received sequence of messages (step 2 of the protocol). A priori this requires time exponential in the number of elapsed rounds, but we will show that it can be done in time exponential in the amount of "current error." Furthermore the frequency with which changes of various distances are made falls off exponentially in the distance, and by ensuring that this exponential dominates the previous one, we achieve constant expected computation time per round.

The idea of such a probabilistically constrained search was introduced in the work of Wozencraft, Reiffen, Fano and others [34, 24, 7] on sequential decoding.

We allot constant time to elementary operations such as pointer following and integer arithmetic; in more conservative models the time alloted would depend on the size of the database and the size of the integers, but in any standard model our time analysis would be affected by a factor of no more than $\log T$.

Some attention to data structures is necessary in the analysis.

As is common in the algorithms literature, we use the term *suffix* to indicate any terminal interval $g_j...g_i$ of a sequence $g_1...g_i$ $(1 \leq j \leq i)$.

The procedure for determining the new guess is as follows. If the newly received character matches that on one of the arcs exiting the current guessed state, then extend the guess along that arc; otherwise extend it arbitrarily along one of the 12 arcs. Now (in either case) determine the longest suffix of this interim guess, such that the fraction of tree code characters in the suffix which differ from those received over the channel, is at least 1/4.

Observe that if this fraction is less than 1/4 in all suffixes then, since every pair of branches differ in half their characters, it is certain that the current guess is the desired (i.e. minimum distance) one. Moreover, for the same reason, if for all suffixes beyond some length ℓ, the fractions are less than 1/4, then we need not examine branches that diverged from our current guess more than ℓ rounds ago. Therefore, if there exist suffixes which violate the 1/4 condition, we find the longest such suffix, say of length ℓ, and determine our new guess as that state which minimizes the distance to the received sequence, among all states which agree with the interim guess in all but the last ℓ rounds. This computation is performed by exhaustively considering all such states.

We now describe how to implement the above computations efficiently. Let the current guess be $g = g_1...g_t$ and let the received sequence be $Z = Z_1...Z_t$. Let $\varepsilon_i = 0$ $\forall i \leq 0$, and for $i > 0$, let $\varepsilon_i = 1$ if $w(g_1...g_i) \neq Z_i$, otherwise $\varepsilon_i = 0$. Now define $\psi(i) = \sum_{j=1}^{i} 5\varepsilon_j - 1$ for $i \geq 1$; and by extension, $\psi(i) = -i$ for $i \leq 0$.

We maintain the following data: for every integer r for which there exists an integer $j \geq 0$ s.t. $\psi(j) = r$, we maintain the pair (r, J_r). Here J_r is a doubly linked list consisting of all integers j for which $\psi(j) = r$, arranged in order of increasing size. We also maintain external pointers to the smallest and largest elements of J_r. Note that if $r < 0$, J_r contains the list $(j_1, ..., j_k)$, while if $r \geq 0$, J_r contains the list $(-r, j_1, ..., j_k)$. Let $m(r)$ be the least integer in J_r.

Observe that at time t, the suffixes of g in which the relative distance is at least 1/4, are precisely those which begin at a time t' for which $\psi(t') \leq \psi(t)$. Since ψ decreases by at most 1 per round, the greatest such suffix begins at time $m(\psi(t))$.

The records (r, J_r) are themselves stored in a doubly linked list, sorted by r.

The procedure is now implemented as follows. At time t, after the previous guess has been extended by one arc, we determine $\psi(t)$ for the new guess g by simply adding 4 or -1 to $\psi(t-1)$. Now if no entry $(\psi(t), J_{\psi(t)})$ exists in the database, a new one is created (with $J_{\psi(t)}$ being either (t) or $(-\psi(t), t)$ depending on the sign of $\psi(t)$). If an entry $(\psi(t), J_{\psi(t)})$ does exist in the database, we append t to the end of the list.

Note that since at time $t-1$ we already had a pointer to the record $(\psi(t-1), J_{\psi(t-1)})$, a constant number of steps suffice to locate the record $(\psi(t), J_{\psi(t)})$, or determine its absence and insert it.

Now if $m(\psi(t)) = t$ then there is no suffix of the current guess in which the distance between g and Z is at least a quarter of the length of the suffix, and we are done with this round. If $m(\psi(t)) < t$ then the greatest such suffix begins at $m(\psi(t))$, and we proceed to exhaustively compute the new suffix which minimizes the distance to Z. This dictates a new guess g'. We update the data structure by first, deleting the entries corresponding to the suffix of g (in reverse order, so the first entry to go is that corresponding to g_t); and second, inserting the entries corresponding to g' (in the usual order, starting with $g'_{m(\psi(t))+1}$).

Let $L(t)$ be the length of the recomputed suffix, $L(t) = t - m(\psi(t))$. The amount of time expended on computation is proportional to $\sum_{t=1}^{5T} 12^{L(t)} + L(t)$. The expectation of this quantity (allowing for all possible ways in which the erroneous messages can arrive, and using lemma 3) is $\sum_t E(12^{L(t)} + L(t)) \leq \sum_t \sum_{L \geq 1} 2^L (2^{-208} 3^{-40})^{L/4} (12^L + L) = O(T)$.

5 Binary Erasure Channel with Feedback

We have focused in this paper on the binary symmetric channel (BSC) as representative of channels with random noise. Some justification for using the BSC is to be found in the fact that the method developed in order to solve the case of the BSC, extended with hardly any modification to the extreme case of an "adversarial" channel. There are however even "tamer" models of error than the BSC; we consider one of these here. In this case we obtain relatively simple and elegant results; but the method involved is not useful for more difficult error models.

We focus in this section upon protocols in which only one processor transmits at any time; as opposed to the work in previous sections, in which both processors were allowed to transmit simultaneously.

Consider the binary erasure channel (BEC). This is a channel with two inputs $0, 1$, and three outputs $0, \mathrm{Err}, 1$. For some p, inputs are transformed into Err with probability p; transmitted successfully with probability $1-p$; and never transformed into the opposite character.

Let us assume the BEC is additionally equipped with perfect feedback, which means that the transmitter can see every character as it arrives at the receiver. (Call this a BECF channel.) Consider the following channel-specific complexity for a communication problem f: the maximum over distributions on input pairs to f, of the minimum over protocols, of the expected number of transmissions until the problem is solved with zero error probability on the BECF. Now define the complexity C_{BECF} of the problem as the product of the previous quantity with the capacity of the channel. Let C^R_{BECF} denote the corresponding quantity on a specified distribution R on input pairs.

Recall the *distributional* complexity D for noiseless channels [35], which is defined as the maximum over distributions on input pairs, of the minimum over protocols, of the expected number of transmissions until the problem is solved. Let D^R denote the corresponding quantity on a specified distribution R on input pairs.

The following result is a zero-error or "Las Vegas" analogue of Shannon's theorem and its converse (and incidentally shows that the definition of C_{BECF} is meaningful).

Theorem 5 *The C_{BECF} complexity of a communication problem f is equal to its distributional complexity D.*

Proof: First we show that $C_{BECF}(f) \leq D(f)$. The capacity of the BECF (as well as the BEC) is $1-p$. Observe that due to the perfect feedback, both processors are always aware of the entire history of their communication session. Fix any distribution $R = \{r(z)\}$ on input pairs. Now, simulate a noiseless-channel protocol on the BECF by simply repeating every transmission until it is received successfully. The length of the BECF protocol on a particular input pair is the sum, over all noiseless steps, of the number of BECF

transmissions required to get that noiseless step across; the expectation of the length of each of these steps is $\frac{1}{1-p}$. Thus an input pair solved in n steps by the noiseless protocol will be solved in an expected $n\frac{1}{1-p}$ transmissions on the BECF. Hence the complexity of the problem is preserved on every input pair, and in particular the average over the distribution R is preserved.

Next we show that $C_{BECF}(f) \geq D(f)$. Fix any distribution $R = \{r(z)\}$ on input pairs. Pick a BECF protocol π achieving complexity $C^R_{BECF}(f)+\epsilon$ for some $\epsilon \geq 0$, smaller than any positive $r(z)$. Now represent π with a binary tree whose root v_0 corresponds to the start of the protocol, while the children of a vertex v correspond to the possible states of the protocol after an additional transmission. Denote by v^S the vertex the protocol proceeds to if the transmission was successful, and v^{Err} the next vertex if the transmission was unsuccessful. Define the following functions at each vertex:

$$h_{\pi,z}(v) = E(\text{ number of remaining transmissions until } \pi \text{ terminates on input pair } z)$$

and

$$h_\pi(v) = \sum_z r(z) h_{\pi,z}(v).$$

Observe that $h_\pi(v_0) = C_{BECF}(f) + \epsilon$.

The performance of the channel is such that for any π, z and v, $h_{\pi,z}(v) = 1+(1-p)h_{\pi,z}(v^S)+ph_{\pi,z}(v^{Err})$. Therefore also

$$(1) \quad h_\pi(v) = 1 + (1-p)h_\pi(v^S) + ph_\pi(v^{Err}).$$

We claim that we may as well assume that $h_\pi(v) = h_\pi(v^{Err})$ for any vertex v of the tree. Suppose this is false at some v. Consider the chain of vertices descended from v strictly through errors: i.e. v^{Err}, $(v^{Err})^{Err}$ (which we may write v^{Err^2}), etc. Exactly the same information is available to the processors at all these vertices, and so π may be edited by replacing the subtree rooted at the "success" child of v (which we denote v^S), by that rooted at the "success" child of any other vertex v^{Err^k} on this chain. (This includes replacing the protocol's action on each input at v, by that specified at v^{Err^k}.) Such a replacement may be made not only at v, but at any vertex v^{Err^i} on the chain. Using (1) we find that

$$h_\pi(v) = \sum_{i=0}^{\infty} p^i((1-p)h_\pi(v^{Err^iS}) + 1)$$

(where v^{Err^iS} is the "success" child of v^{Err^i}). If there is a vertex v^{Err^k} on the error-chain descended from v, whose "success" child v^{Err^kS} attains the infimum among all the values $\{h_\pi(v^{Err^iS})\}_{i=0}^{\infty}$, then we may reduce $h_\pi(v)$ by replacing all of the "success" children $\{v^{Err^iS}\}$ by v^{Err^kS}. Even if the infimum is not achieved by any v^{Err^kS}, $h_\pi(v)$ can be reduced by choosing a v^{Err^kS} with $h_\pi(v^{Err^kS})$ sufficiently close to the infimum, and making the same replacement.

We transform π into a protocol π' satisfying $h_{\pi'}(v) = h_{\pi'}(v^{Err})$ at all its vertices, by first editing π in this way at the root, then editing the resulting protocol at the vertices one level away from the root, and so forth. (This process is infinite but it gives a well-defined π'.) Note that $C^R_{BECF}(f) \leq h_{\pi'}(v_0) \leq C^R_{BECF}(f) + \epsilon$.

From 1 and the fact that $h_{\pi'}(v) = h_{\pi'}(v^{Err})$, we now have at every vertex:

$$h_{\pi'}(v) = 1 + (1-p)h_{\pi'}(v^S) + ph_{\pi'}(v).$$

Therefore $h_{\pi'}(v^S) = h_{\pi'}(v) - \frac{1}{1-p}$.

Now we use π' to solve f on a noiseless channel. Since no errors occur, the protocol simply proceeds to "success" children in every transmission. After $(1-p)C^R_{BECF}(f)$ levels of descent, we will reach a vertex u such that $h_{\pi'}(u) \leq \epsilon$. If there were any input pair z such that π' had not yet terminated on z at u, then $h_{\pi',z}(u)$ would have had to be at least 1, and therefore $h_{\pi'}(u)$ would have had to be at least $r(z)$. However

this is a contradiction by our assumption on ϵ. Therefore π' terminates on all input pairs by the time it reaches u. The theorem follows because u is at level $(1-p)C^R_{BECF}(f)$. □

It is evident from this section that the coding problem for interactive communication, is much simpler on the BECF than on more general channels, even among discrete memoryless channels. This is a phenomenon familiar already from data transmission, where the BECF model and variants of it have been studied in work beginning with Berlekamp [1].

6 Discussion

Insofar as interactive protocols model the operation of a computer whose inputs and processing power are not localized, this paper may be regarded as presenting a coding theorem for computation.

Our theorem suffers, however, a drawback similar to one which afflicted Shannon's original work: namely that a good code has only been shown to exist, and has not been explicitly exhibited. From a practical standpoint this was not necessarily a severe problem for data transmission, since a randomly constructed code almost surely has good properties. Even so this was hardly satisfactory. Explicit codes achieving arbitrarily low error at a positive asymptotic rate were not exhibited until Justesen's work of 1972 [15]. This drawback is even greater for implementation of the present work, since the existence arguments for the required tree codes do not provide one with high probability.

Explicit construction of a tree code can reasonably be interpreted as meaning that each label must be computable in time polynomial in the depth of the tree. (Or, for an infinite tree, polynomial in the depth of the label.) The current state of knowledge on this very interesting problem is as follows. First, a construction is known with alphabet size polynomial in the depth of the tree [5]. Second, if the definition of a tree code is relaxed so that the Hamming distance condition is required only for pairs of vertices whose distance from their least common ancestor is at most logarithmic in the depth of the tree, then the existence proof can be adapted to provide an explicit construction. These constructions provide, respectively, an explicit protocol with logarithmic communication overhead and exponentially small probability of error; or, with constant communication overhead and polynomially small probability of error.

A word on the constant $2^{-208}3^{-40}$ in lemma 2. External events will intervene long before the first interesting event happens in an implementation of the protocol, if the error probability in each "track" transmission is truly amplified to this degree. Correspondingly, the rate achieved by the protocol run with this parameter, although positive, will be absurdly low. However this appears to be primarily an artifact of the analysis and not of the protocol; and we have endeavored to make the presentation as simple as possible at the expense of optimization of the parameters. In practice an optimized version of the protocol will likely achieve higher rate than provided in the bounds.

The present work shows that interactive protocols can be deterministically simulated on a BSC at a rate within a constant factor of channel capacity, but in the data transmission case it is possible to communicate as close to capacity as is desired. In the view of the author it is very likely that there is indeed a multiplicative gap between the channel capacity, and the rate which can always be guaranteed in the interactive case. It appears that (much as for the quantity R_0 in the sequential decoding literature) a certain amount of "backtracking" has to be allowed for. It would be of great interest to demonstrate this gap, or else close it by providing a better simulation.

Even if the above is true, this does not suggest that there will not be problems which cannot be simulated with relatively greater efficiency in the noisy case. Shannon's theorem is accompanied by its converse: that at any transmission rate beyond channel capacity, a code must suffer high error probability on at least some codewords. Since n-bit transmissions are a special case of n-bit noiseless communication protocols, this provides a restricted sort of converse for the coding theorem for interactive protocols. However, this does not mean that any particular noiseless-channel protocol does not in fact have a much more efficient

noisy-channel protocol (after factoring in the channel capacity). For instance, it may be that a new protocol can be devised for the case of noisy channels, that is not at all a simulation of a noiseless-channel protocol, and that takes advantage of the greater number of (albeit noisy) rounds available. Furthermore, it may be that the best noiseless protocol does not have use for the two bits being exchanged in each round in our model, but only one of them; while they might be of use to a noisy protocol. In this regard one may also want to keep in mind the work of Shannon on two-way channels with a joint constraint on capacity [30].

In the above context, we mention that the use of randomness as a resource must be considered carefully. If it is allowed, then one should compare randomized complexities in both the noiseless and noisy settings. However if a comparison of deterministic complexities is preferred, then one must be wary of the processors gaining random bits from the noise on the channel. (For instance, to avert this, an adversary might be allowed some control over the error probability of the channel in each transmission.)

Theorem 4 indicates that a coding theorem can be obtained for most channels of interest, but it would of course be desirable to study what rate can be obtained on various channels. We should note however that a complicating factor in this question, is that the capacity of a channel with memory, unlike that of one without memory, can be increased by feedback. Hence there may be competition for use of each direction of the channel, between its role in transmitting messages in that direction, and its role in amplifying the capacity of the the channel in the opposing direction.

In earlier work of the author in this area it was proposed that the problem of interactive communication in the presence of noise, also be studied for networks of more than two processors [26]. In particular one would ask to what extent the coding theorem might be extended to the efficient simulation of noiseless distributed protocols, on networks with noise. This question has been answered by Rajagopalan and the author in a forthcoming publication.

We draw attention also to the work of Gallager [13], who has previously considered a different problem concerning noise in a distributed computation. He considered a complete network of n processors, each of which in a single transmission can broadcast one bit, which arrives at each of the other processors subject to independent noise. He studied a specific problem on this network: supposing that each processor receives a single input bit, he showed how to quickly and reliably compute the combined parity of all the inputs. (There is as yet a gap between the upper and lower bounds in this interesting problem, however.)

Karchmer and Wigderson [16] observed a certain equivalence between communication complexity and circuit complexity, and thereby stimulated great recent interest in communication complexity. While noisy circuits have been studied in the literature (e.g. [33, 21, 23, 22, 2, 3, 8, 11, 25, 6]), (as have noisy cellular automata, [31, 9, 10]) the correspondence between circuits and communication protocols does not appear to extend to the noisy cases of each. Elias [4] and later Peterson and Rabin [20] investigated the possibility of encoding data for computation on noisy gates, and the extent to which these gates might be said to have a finite (nonzero) capacity for computation. (Here, as for channel transmission, noiseless encoding and decoding of the data before and after the computation are allowed.)

Acknowledgments

This research was conducted at MIT (see [26, 27] for preliminary presentations and related work) and at U. C. Berkeley (see [28]). Thanks to my advisor Mike Sipser whose oversight and encouragement in the early stages of this work were invaluable. For consultations and comments thanks also to Dan Abramovich, Manuel Blum, Peter Elias, Will Evans, Robert Gallager, Wayne Goddard, Oded Goldreich, Mauricio Karchmer, Richard Karp, Claire Kenyon, Dan Kleitman, Mike Klugerman, Nati Linial, Mike Luby, Moni Naor, Yuval Peres, Nicholas Pippenger, Yuval Rabani, Sridhar Rajagopalan, Andrew Sutherland, Umesh Vazirani, Boban Velickovic and David Zuckerman. Thanks to the editor, Rene Cruz, and the anonymous referees, whose careful comments substantially improved the paper.

Funding at MIT was provided by an ONR graduate fellowship, an MIT Applied Mathematics graduate fellowship, and grants NSF 8912586 CCR and AFOSR 89-0271. Funding at Berkeley was provided by an NSF postdoctoral fellowship.

Current address: College of Computing, Georgia Institute of Technology, Atlanta GA 30332-0280, USA.

References

[1] E. R. Berlekamp. Block coding for the binary symmetric channel with noiseless, delayless feedback. In H. B. Mann, editor, *Error Correcting Codes*, pages 61–85. Wiley, 1968.

[2] R. L. Dobrushin and S. I. Ortyukov. Lower bound for the redundancy of self-correcting arrangements of unreliable functional elements. *Prob. Inf. Trans.*, 13:59–65, 1977.

[3] R. L. Dobrushin and S. I. Ortyukov. Upper bound for the redundancy of self-correcting arrangements of unreliable functional elements. *Prob. Inf. Trans.*, 13:203–218, 1977.

[4] P. Elias. Computation in the presence of noise. *IBM Journal of Research and Development*, 2(4):346–353, October 1958.

[5] W. Evans, M. Klugerman, and L. J. Schulman. Constructive tree codes with polynomial size alphabet. Manuscript.

[6] W. Evans and L. J. Schulman. Signal propagation, with application to a lower bound on the depth of noisy formulas. In *Proceedings of the 34th Annual Symposium on Foundations of Computer Science*, pages 594–603, 1993.

[7] R. M. Fano. A heuristic discussion of probabilistic decoding. *IEEE Transactions on Information Theory*, pages 64–74, 1963.

[8] T. Feder. Reliable computation by networks in the presence of noise. *IEEE Transactions on Information Theory*, 35(3):569–571, May 1989.

[9] P. Gács. Reliable computation with cellular automata. *J. Computer and System Sciences*, 32:15–78, 1986.

[10] P. Gács and J. Reif. A simple three-dimensional real-time reliable cellular array. *J. Computer and System Sciences*, 36:125–147, 1988.

[11] A. Gál. Lower bounds for the complexity of reliable boolean circuits with noisy gates. In *Proceedings of the 32nd Annual Symposium on Foundations of Computer Science*, pages 594–601, 1991.

[12] R. G. Gallager. *Information Theory and Reliable Communication.* Wiley, 1968.

[13] R. G. Gallager. Finding parity in a simple broadcast network. *IEEE Trans. Inform. Theory*, 34(2):176–180, March 1988.

[14] G. H. Hardy and E. M. Wright. *An Introduction to the Theory of Numbers.* Oxford, fifth edition, 1979.

[15] J. Justesen. A class of constructive, asymptotically good algebraic codes. *IEEE Transactions on Information Theory*, IT-18:652–656, September 1972.

[16] M. Karchmer and A. Wigderson. Monotone circuits for connectivity require super-logarithmic depth. In *Proceedings of the 20th Annual Symposium on Theory of Computing*, pages 539–550, 1988.

[17] Lipton and Sedgewick. Lower bounds for VLSI. In *Proceedings of the 13th Annual Symposium on Theory of Computing*, pages 300–307, 1981.

[18] L. Lovász. Communication complexity: A survey. In Korde et al, editor, *Algorithms and Combinatorics*. Springer-Verlag, 1990.

[19] C. H. Papadimitriou and M. Sipser. Communication complexity. In *Proceedings of the 14th Annual Symposium on Theory of Computing*, pages 196–200, 1982.

[20] W. W. Peterson and M. O. Rabin. On codes for checking logical operations. *IBM Journal of Research and Development*, 3:163–168, April 1959.

[21] N. Pippenger. On networks of noisy gates. In *Proceedings of the 26th Annual Symposium on Foundations of Computer Science*, pages 30–36, 1985.

[22] N. Pippenger. Reliable computation by formulas in the presence of noise. *IEEE Transactions on Information Theory*, 34(2):194–197, March 1988.

[23] N. Pippenger. Invariance of complexity measures for networks with unreliable gates. *J. ACM*, 36:531–539, 1989.

[24] B. Reiffen. Sequential encoding and decoding for the discrete memoryless channel. *Res. Lab. of Electronics, M.I.T. Technical Report*, 374, 1960.

[25] R. Reischuk and B. Schmeltz. Reliable computation with noisy circuits and decision trees — a general $n \log n$ lower bound. In *Proceedings of the 32nd Annual Symposium on Foundations of Computer Science*, pages 602–611, 1991.

[26] L. J. Schulman. *Communication in the Presence of Noise*. PhD thesis, Massachusetts Institute of Technology, 1992.

[27] L. J. Schulman. Communication on noisy channels: A coding theorem for computation. In *Proceedings of the 33rd Annual Symposium on Foundations of Computer Science*, pages 724–733, 1992.

[28] L. J. Schulman. Deterministic coding for interactive communication. In *Proceedings of the 25th Annual Symposium on Theory of Computing*, pages 747–756, 1993.

[29] C. E. Shannon. A mathematical theory of communication. *Bell System Tech. J.*, 27:379–423; 623–656, 1948.

[30] C. E. Shannon. Two-way communication channels. *Proc. 4th Berkeley Symp. Math. Stat. and Prob. (reprinted in Key Papers in Information Theory, D. Slepian, ed., IEEE Press, 1974)*, 1:611–644, 1961.

[31] M. C. Taylor. Reliable information storage in memories designed from unreliable components. *Bell System Tech. J.*, 47(10):2299–2337, 1968.

[32] C. D. Thompson. Area-time complexity for VLSI. In *Proceedings of the 11th Annual Symposium on Theory of Computing*, pages 81–88, 1979.

[33] J. von Neumann. Probabilistic logics and the synthesis of reliable organisms from unreliable components. In C. E. Shannon and J. McCarthy, editors, *Automata Studies*, pages 43–98. Princeton University Press, 1956.

[34] J. M. Wozencraft. Sequential decoding for reliable communications. *Res. Lab. of Electronics, M.I.T. Technical Report*, 325, 1957.

[35] A. C. Yao. Probabilistic computations: Toward a unified measure of complexity. In *Proceedings of the 18th Annual Symposium on Foundations of Computer Science*, pages 222–227, 1977.

[36] A. C. Yao. Some complexity questions related to distributive computing. In *Proceedings of the 11th Annual Symposium on Theory of Computing*, pages 209–213, 1979.

[37] A. C. Yao. The entropic limitations on VLSI computations. In *Proceedings of the 13th Annual Symposium on Theory of Computing*, pages 308–311, 1981.