A Coding Theorem for Distributed Computation

Sridhar Rajagopalan* Leonard J. Schulman†
University of California, Berkeley.

Abstract

Shannon's Coding Theorem shows that in order to reliably transmit a message of T bits over a noisy communication channel, only a constant slowdown factor is necessary in the case when the channel is noisy, relative to the case in which the channel is noiseless. (The time required is asymptotically $T\frac{1}{C}$, where $0 < C \leq 1$ is the "Shannon capacity", a function only of the noise characteristics.) The theorem ensures that the probability of a decoding error is exponentially small in the message length T. Recently the second author obtained an analogous result for arbitrary interactive communication protocols between two processors.

In the present paper we provide a coding theorem for all distributed protocols on static topology networks. We prove that any protocol which runs in time T on a network of degree d having noiseless communication channels, can, if the channels are in fact noisy (each a binary symmetric channel of capacity C), be simulated on that network in time proportional to $T\frac{1}{C}\log(d+1)$. The probability of failure of the protocol is exponentially small in T.

We also provide specialized results for the problem of broadcasting data over a noisy chain of processors.

1 Introduction

This paper is concerned with the question of whether it is possible to sustain distributed computation in the presence of noise; and if so, at what cost in efficiency and reliability.

Shannon in his classical coding theorem showed that data can successfully and efficiently be transmitted in a noisy environment [27]. The outstanding features of the theorem are first, that if a large block of data is to be transmitted over a noisy channel, the transmission need slow down by only a constant factor (relative to the noiseless case) and second, that one can achieve an error probability exponentially small in the length of the block of data.

Data transmission is an elementary step in computation: this is overt in distributed or parallel computation, but is also true of sequential computers, where almost invariably, several logical units are involved.

This leads to the question: if a computation is being performed by several processors linked in a network, and if the communications in this network are unreliable, what is the effect on the efficiency and reliability with which this computation can be performed?

In the main result of the present paper, we address this question by providing a network analog of Shannon's coding theorem. We show that a distributed protocol which runs on a network of degree d can be simulated, if the links of that network are noisy, with a slow-down factor proportional to $\log(d+1)$ and with probability of error exponentially small in the length of the original protocol.

In this we extend a recent result of Schulman[26] in which such a coding theorem was shown for any interactive protocol involving two processors.

The physical context for this work is simply that noise is an inherent characteristic of physical systems; and, in particular, of communications hardware. As computing systems grow, so does the necessity of dealing with such noise either by "overbuilding" the hardware to make it error-free; by introducing redundant hardware; or by implementing error-correction in software.

In contrast to physical computers, machine-based models of computing typically take noise-free devices as their basis: e.g. Turing machines, cellular automata, formal circuits, and parallel and distributed systems. Much work has been driven by the need to bridge this gap between model and reality. The first investigations in this direction were by Shannon in 1948 [27], for the problem of data transmission over noisy links; and by von Neumann in 1956 [34], for the problem of computation by circuits with unreliable components.

This paper is a further contribution toward bridging the gap between noiseless models, and noisy reality. The model we address is closest to a distributed system (or a single computer with several processors). In this case, the individual processors are computationally macroscopic; correspondingly in our model the processors are capable of extensive computing. Further, we assume that all noise in our system occurs on the communication links between the processors, and not in their internal computations. (This is a useful model because the error rate on a long communication link may be significantly worse than that inside a localized processor.)

1.1 Background

We very briefly review some of the extensive literature relating noisy and noiseless models. The foundational results of both Shannon and von Neumann were essentially positive statements. Shannon in his foundational work described the

*Supported by NSF grants # IRI 91-20074 and CCR-9310214. Email: sridhar@cs.Berkeley.EDU.

†Supported by an NSF Postdoctoral Fellowship. Email: schulman@cs.Berkeley.EDU.

Proc. 26'th STOC (ACM Symp. Theory Comp.), ACM Press, 1994.

notion of the *capacity* of a communication link, and showed that data can be transmitted with high reliability across the link at any rate below capacity. He also showed a converse to this coding theorem, namely that communication fails at any rate above capacity; but the striking technological implication lay in the coding theorem itself, which showed that "modulo coding theory", one may as well think of data transmission in a noise-free model. The Shannon theory applies to a wide class of channels; its essential characteristics however are already present in the case of a *binary symmetric* channel, one in which in each transmission, independently of all prior events, the transmitted bit is flipped with fixed probability $\epsilon < 1/2$. Therefore in this paper we will focus on this noise model.

Von Neumann's work, in conjunction with later work by Dobrushin and Ortyukov [3, 4], Pippenger [17], Pippenger Stamoulis and Tsitsiklis [20], Gal [9] and Reischuk and Schmeltz [23], shows that a logarithmic factor overhead in size is always sufficient, and sometimes necessary, for the simulation of noiseless by noisy circuits. (Here it is understood that each circuit component malfunctions independently with constant probability. Pippenger has also explored more general noise models [19].) Furthermore, von Neumann showed that only a constant factor overhead in depth was needed in going from noiseless to noisy circuits; that a factor greater than 1 was, in fact, sometimes necessary was demonstrated by Pippenger [18], and the form of the lower bound was later improved by Feder [6] and Evans and Schulman [5].

The noise-resilient circuits described by von Neumann and in the subsequent literature, are not embeddable without significant edge dilation in finite dimension. Since longer wires are more likely to malfunction, the question naturally arose of whether computation can persist in the more restrictive model of noisy cellular automata. Affirmative answers in this regard were supplied by Taylor [28], Toom [30, 31, 32], Tsirel'son [33], Gács [7] and Gács and Reif [8].

There is an essential difference between the circuit and cellular automata results, on the one hand, and the studies in information and coding theory on the other. In the former models no component of the system is assumed to be noiseless: an entire reliable computation must be synthesized out of individual unreliable components. In the latter situation computations are assumed to be reliable, but communications are assumed to be error prone. The two suppositions correspond naturally to a fine-grained modeling of a system and a coarse grained modeling of it. Our work is in the latter line: The study of what is feasible when encoding, decoding and other local computation can be implemented cheaply and reliably at each processor in the course of a protocol.

One case of this question was proposed by El Gamal and studied by Gallager [10]. They considered a complete network of noisy links among N processors, and a restricted communication model: in each round, a processor is allowed to broadcast some bit to the entire system. (The noise events on each channel are independent.) Each processor starts the computation with some private bit, and the goal of the computation is for all processors to learn all the bits[1]. The cost of the computation is measured as the total number of broadcasts. The naive protocol for this problem requires $O(N \log N)$ broadcasts, but Gallager showed how to accomplish the computation with only $O(N \log\log N)$ broadcasts. It is a very interesting question whether this is best possible.

We are concerned in this paper with communication as a resource for computation. An essential aspect of this setting is the *interactive* communication that is needed between processors. In contrast, in information theory and coding theory communication has been viewed entirely in terms of the data transmission problem (even in those papers which studied the benefits for data transmission, of feedback from the recipient to the transmitter).

The interactive model for communication complexity, in which the input to a problem is split between two processors linked by a noiseless channel, was introduced by Yao [35]. The model was devised to measure the cost incurred in departing from the single-processor model of computation. It has since been intensively investigated (e.g. [36, 16, 29, 11]; and see [12] for a survey).

The effect of noise upon this model was studied by Schulman [24, 25, 26], who proved an analog of Shannon's coding theorem showing that any interactive protocol for a pair of processors, designed for a noiseless communication link, could be simulated on a noisy link with only constant communication overhead. This result opened the way to the following basic question which was posed in [24, 25]: can the coding theorem for interactive protocols be extended to the efficient simulation of noiseless distributed protocols, on networks with noise?

1.2 Overview of this Paper

We answer the above question by providing a "compiler" for distributed protocols which, given the description of a protocol which runs on a certain network with noiseless links, converts the protocol into one which runs on the same network but with noisy links. Our result is:

Theorem 1.1 *Any protocol which runs in time T on an N-processor network of degree d having noiseless communication channels, can, if the channels are in fact noisy (each a binary symmetric channel of capacity C, $0 < C \leq 1$), be simulated on that network in time $O(T\frac{1}{C}\log(d+1) + \frac{1}{C}\log N)$. The probability of failure of the protocol is $e^{-\Omega(T)}$.*

By comparison, Shannon's original coding theorem states that a block of T bits of data can be transmitted across such a channel in time $O(T\frac{1}{C})$, with error probability $e^{-\Omega(T)}$; and Schulman's paper describes a "compiler" for arbitrary noiseless communication protocols of length T between two processors, which guarantees a running time of $O(T\frac{1}{C})$ and error probability $e^{-\Omega(T)}$.

There are two principal ideas used in obtaining our results. The first is that in our protocol, errors in the system that have a "space-like" separation – that occur closer in time than the distance between them in the network – cause no more delay to the protocol than a single error. This allows us to make an argument of a type that has come to be known as a 'delay sequence argument'. This type of argument has been successfully employed in various contexts: for instance by Aleliunas [1], Luby [13], and Martel, Subramonian and Park [14] in parallel communications; and by Ranade in routing [22] and load balancing [21]. The second is a coding technique, tree codes, introduced in [26].

Two major difficulties remain concerning theorem 1.1. The first is that, while there is an existence proof for the

[1] Actually the stated goal was just to compute the parity of the bits, but Gallager's protocol as well as others we are aware of, solve the harder problem with the same effort, and no lower bound is known to separate the two problems.

type of codes employed in our proof – the "tree codes" – no explicit construction of such codes is known. The second is that, even given an explicit tree code, the computational overhead involved in certain decoding steps of our protocol, is exponential in T. Both of these difficulties parallel those which remained following the original proof of Shannon's coding theorem.

There is one very special case of the general network problem, however, in which we can provide a protocol which is not burdened by these computational difficulties. The problem is this: transmit B bits from end to end of a chain of N processors. Of course in the absence of noise this can be accomplished in time $N + B - 2$. The network here is as simple as can be, and so is the problem statement, just data transmission. Remarkably, no time-efficient solution seems to have been provided for this problem until very recently when Nisan and Parnas [15] analyzed the case in which the processors are connected by binary erasure channels. (Channels which sometimes output "Error" but never the wrong bit.) Specifically, Nisan and Parnas showed that if the bits are "pipelined" through the chain, then except for an event of probability $e^{-\Omega(N+B)}$, the last bit will arrive at the end of the chain within time $O(N + B)$. We address the case in which (as in the rest of our paper) the channels are binary symmetric. In this case, even the problem of efficiently transmitting a single bit ($B = 1$) from end to end appears to be not entirely trivial.

Let the probability of error in the channels be $(1 - \delta)/2$ for some $0 < \delta \leq 1$. As indicated, our first result concerning the chain broadcast problem is an efficient protocol:

Theorem 1.2 *A B bit message can be transmitted from end to end of an array of N processors interconnected by binary symmetric channels in time $O((B + N) \cdot e^{O(\log^*(B+N))})$. All computations necessary for the protocol are polynomial time in $B + N$. The probability of an error in transmission is at most $e^{-\Omega(B+N)}$.*

Our second result concerning this problem is a lower bound showing that the noise on the channels does force the broadcast to be slower than it would be in the absence of noise:

Theorem 1.3 *Suppose that each input bit $0, 1$ is equally likely. Then for any $\gamma < 1$, for sufficiently large values of n, any bit which is computed at processor n at time $(n - 1)\gamma/\delta + 1$ can be equal to the input bit with probability at most $1/2 + o(1)$.*

2 The Network Model

Consider a network $\mathcal{N}$ with processor set V and interconnections E executing a synchronous distributed protocol Π. We use the letter d to denote the maximum degree of any vertex. In Π each processor repeats the following three steps in sequence in each time step:

- Receive a bit on each incoming link.
- Do a local computation.
- Transmit a bit on each outgoing link.

The bit transmitted by processor p on link (p, q) at the end of time step t is received at processor q at the inception of time step $t + 1$.

Formally a protocol Π on a network $\mathcal{N}$ is a collection of functions $\{\Pi^p\}$, one for each processor p in the network. Let $\mathrm{indeg}(p)$ and $\mathrm{outdeg}(p)$ denote the in-degree and out-degree of processor p. (How many processors can send messages to, and receive messages from, p. Typically the communication links are bidirectional but we do not require this.) Each function Π^p maps $\mathrm{indeg}(p)$ binary strings $\alpha^p_1(t), ..., \alpha^p_{\mathrm{indeg}(p)}(t)$, each of length t, to an $\mathrm{outdeg}(p)$-tuple of bits:

$$\Pi^p(t) = (\Pi^p_1(t), ..., \Pi^p_{\mathrm{outdeg}(p)}(t))$$

Each string $\alpha^p_i(t)$ is the history of all bits that have been received at processor p by time t along the i'th incoming edge to p. Each bit $\Pi^p_j(t)$ is the response of the processor along it's j'th outgoing edge.

3 The Simulation Protocol

In this section we describe a simulation protocol Σ which turns an arbitrary network protocol Π, designed for use on a noiseless network, into one which runs successfully with high probability even if the channels are noisy.

3.1 Communication using Tree-Coded Channels

In Σ, as we will describe it, processors will use three symbols, $\{\mathtt{0}, \mathtt{1}, \mathtt{bkp}\}$, to communicate with each other. Two steps will be required to turn such a symbol into a binary string for transmission on a channel:

$\mathcal{T}$: Encoding of the symbol using a ternary tree code.

χ: Encoding of the tree code letter into a binary string, using a code for data transmission.

We will speak abstractly of a tree-coded channel as one in which transmissions are encoded using these two steps, and decoded using a matching two step process.

In order to explain this process we briefly describe tree codes. For more detail on this subject see [26]. A ternary tree code is a ternary tree (each internal node has 3 children) whose edges are labelled with characters chosen from S, a finite alphabet. To encode a string $x \in \{\mathtt{0}, \mathtt{1}, \mathtt{bkp}\}^*$ of length t using a tree code $\mathcal{T}$, we set $\mathcal{T}(x)$ to be the concatenation of the characters found on the simple path from the root, to the vertex of the tree representing x.

If $s = (s_1...s_m)$ and $r = (r_1...r_m)$ are words of the same length over S, say that the distance $D(s, r)$ between s and r is the number of positions i in which $s_i \neq r_i$ (Hamming distance). A ternary tree of depth n is a rooted tree in which every internal node has 3 children, and every leaf is at depth n (the root is at depth 0).

Definition 3.1.1 *A ternary tree code over alphabet S and of depth n is a ternary 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, and whose least common ancestor is at some depth $h-l$. 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 $D(W(v_1), W(v_2)) \geq l/3$. (See figure 1).*

The key fact regarding tree codes is:

Lemma 3.1.2 ([26]) *There is a finite s such that an alphabet S of size s suffices to label a ternary tree code of any depth.*

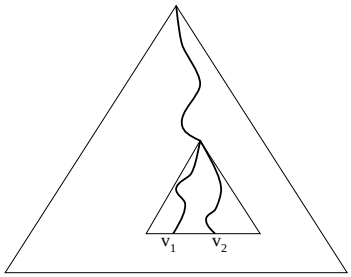

Figure 1: The figure exhibits the distinguishing property of tree codes. The strings v_1 and v_2 may differ very little, but their tree code representations differ in $l/3$ character positions where l is the height of the subtree pictured here.

Thus the first step of the encoding described above, is accomplished as follows: let $x \in \{0, 1, \mathtt{bkp}\}^*$ be the past history of transmissions from processor p to processor q, and suppose that processor p now wishes to send a character $a \in \{0, 1, \mathtt{bkp}\}$. Then the product of this step of the encoding is the letter of S which is inscribed on the arc between the vertices $\mathcal{T}(x)$ and $\mathcal{T}(xa)$.

In the second step of the encoding, the letter of S is encoded as a binary string using a data transmission code $\chi : S \to \{0, 1\}^k$. We will refer to the parameter k as the *dilation* of the tree-coded channel. (We will later specify the range for k that is useful for our purpose.) Finally this resulting string is transmitted across the binary noisy channel.

Decoding of a tree-coded channel is accomplished as follows:

χ^{-1}: Decode the received k-bit string into a letter of S using the decoding algorithm for the data transmission code.

$\mathcal{T}^{-1}$: Decode the tree code using the following maximum likelihood criterion: If $\hat{y}$ is the received tree code string (the concatenation of all letters of S decoded using χ^{-1} in the present and previous rounds) then decode to the string $\hat{x} \doteq \mathcal{T}^{-1}(\hat{y})$ which maximizes $P(\hat{y}|x)$ among all strings $x \in \{0, 1, \mathtt{bkp}\}^*$.

Define an *edge character error* to be the event that a tree code character is corrupted during transmission across an edge, i.e. that for a transmitted character $a \in S$, the noise is such that $\chi^{-1}(\chi(a) + \text{noise}) \neq a$. Observe that the probability of an edge character error decreases exponentially in k. Say also that a *processor character error* (which we will occasionally refer to simply as a character error) occurs at processor p at time t if p makes an edge character error in decoding any of the characters it has received in that round from its various neighbors.

In addition to the character errors, there is another type of error we need to consider. Whether or not p makes a character error in decoding the character received from a neighbor q in the current round, it may make an error in $\mathcal{T}^{-1}$. Often these events will occur together, but they need not: the tree code may be incorrectly decoded in spite of correct reception of the current character, because of the lingering effect of previous incorrect receptions; while the tree code may be correctly decoded in spite of a current incorrect reception, because the map from the tree code alphabet to $\{0, 1, \mathtt{bkp}\}$ is not injective. If p makes an error in $\mathcal{T}^{-1}$ in round t for the message from one of its neighbors, we will say that an *edge tree code error* has occurred on that edge at time t; and if p makes such an error in round t on the message from any of its neighbors, then we will say that a *processor tree code error* (or just a tree code error) has occurred at processor p at time t. Hence if there is no processor tree code error at (p, t), then p has correctly decoded the tree codes for *all* of its neighbors at time t.

Observe that successive edge character errors on an edge (q, p) are almost independent events[2]. On the other hand, successive edge tree code errors are not independent. To the contrary, they are determined by many of the same channel noise events, and so they may be significantly correlated. Understanding the behavior of the protocol in spite of these correlations will require some attention.

3.2 The Protocol

Communications in Σ are executed through the tree coded channel mechanism, and Σ can now be specified entirely in terms of transmissions and receptions of characters in $\{0, 1, \mathtt{bkp}\}$.

Processors running Σ will in every time step either simulate one time step of Π, or erase one previously simulated step of Π. If the processor goes one step forward, then a data bit (0 or 1) is transmitted on on each outgoing channel. If the processor takes a step back, then the bkp character is transmitted on every channel. More formally, if p has outdegree outdeg(p), $\Sigma^p(t)$ is either a binary outdeg(p)-tuple or the outdeg(p)-tuple $(\mathtt{bkp}, \mathtt{bkp} \cdots, \mathtt{bkp})$.

The direction of movement is governed by the notion of consistency which we will describe below. Observe that there is a well defined notion at each processor p and time t of the current time step of Π being simulated. We will refer to this as the *apparent time* $\mathrm{AT}(p, t)$. If $\mathrm{B}(p, t) \stackrel{\text{def}}{=}$ the number of times p has backed up before time t, then $\mathrm{AT}(p, t) \stackrel{\text{def}}{=} t - 2\mathrm{B}(p, t)$.

A string over $\{0, 1, \mathtt{bkp}\}$ is parsed to a string over $\{0, 1\}$ in the following way: every bkp character erases the last preceding unerased $\{0, 1\}$ character (much like the Back Space key on most keyboards)[3]. For instance the string $0, 1, 0, \mathtt{bkp}, 1, 0, 1, \mathtt{bkp}, 1, \mathtt{bkp}, \mathtt{bkp}, 0$ is parsed to $0, 1, 1, 0$. If p has transmitted a sequence $x^{(p,q)} \in \{0, 1, \mathtt{bkp}\}^*$ or received a sequence $\hat{x}^{(q,p)} \in \{0, 1, \mathtt{bkp}\}^*$ then the parsed versions of these strings, $w^{(p,q)}$ and $\hat{w}^{(q,p)}$ in $\{0, 1\}$, will be referred to as the transcripts of p's communications to and from processor q in Π; the collection of these transcripts, for all in- and out-neighbors of processor p, will be referred to as the

[2] They are determined by disjoint, and therefore independent, sequences of noise events on the channel; the only reason the edge character errors are not entirely independent is that the distances among various codewords of χ vary, and so the probabilities of various errors depend slightly on which message is sent. This slight non-independence will not affect the analysis and one may as well think of the edge character errors as independent.

[3] More formally, parsing follows the grammar rule $(0|1)\mathtt{bkp} \mapsto \lambda$ where λ is the empty string and | allows selection. It is easily seen that repeated application of the grammar rule on $\hat{x}^{(q,p)}$ and $x^{(p,q)}$ results in a unique minimum length string over $\{0, 1\}$.

transcript of Π at p. The action taken by p in the succeeding round of Σ will be determined entirely by this transcript (and in particular, not by any further information available in the strings $x^{(p,q)}$ or $\hat{x}^{(q,p)}$)[4]. Notice also, that the length of each outgoing transcript from p at time t is exactly the same and is equal to $\mathrm{AT}(p,t)$.

We will say that the transcript at (p,t) is *consistent* if for every time τ and every out-neighbor q of p, the τ'th character of the output transcript $w^{(p,q)}$ equals the character specified by the protocol Π, given the prefix of the transcript up through time $\tau - 1$.

There are two ways in which the transcript at (p,t) might be inconsistent. The first is that the current transcript, as decoded from the tree codes using max-likelihood decoding, disagrees with the transcript decoded in past rounds. (Because of either past or present decoding errors.) Therefore some of the actions taken in past time steps may be different from what they should have been based upon the current transcript. The second way can occur if one of p's neighbors has backed up in Π. In this case p may have in the past issued some messages of Π, which are now unsubstantiated since the incoming data required to determine them, has been erased or modified. Observe that this second kind of inconsistency can occur even if p has made no decoding errors.

We can now state our protocol.

Simulation Protocol Σ: If the transcript is consistent, transmit the data bits indicated by Π; otherwise back up.

To see that this completely specifies Σ we need only observe that $\mathrm{AT}(p,t) \equiv t \bmod 2$ and therefore a set of data bits is specified by Π unless one the the incoming transcripts is shorter by 2 than $\mathrm{AT}(p,t)$ (the length of p's outgoing transcripts), in which case the transcript is inconsistent and p will choose to back up.

4 Mechanics of the Protocol

In order to understand the protocol we will need to classify the events that can occur at a processor in any round, into several types. The function $\textsc{Action}(p,t)$ which takes as argument a (processor, time) pair, will describe this classification. Note that the classification is one we make for purpose of analysis: it is not one that the processors can determine while they are running the protocol.

Many of the interesting phenomena in our protocol arise already in the case of protocols on a bidirectional linear array, and can therefore be easily illustrated (figures 2, 3). In such an array each processor in each time step exchanges a codeword of χ with each of its two neighbors.

We first consider the possibility that a processor tree code error occurred at processor p at time t. In this case if p decided to back up, we set $\textsc{Action}(p,t) = \mathsf{bx}$. On the other hand if p decided to transmit data of Π, and if any of that data disagrees with the data that would indeed be transmitted in Π at (p,t) on a noiseless network, then we set $\textsc{Action}(p,t) = \mathsf{x}$. Events of type x are depicted in figure 2 at $(6,2),(4,5)$ and $(10,5)$; and in figure 3 at $(11,6)$. An event of type bx is depicted in figure 3 at $(7,4)$.

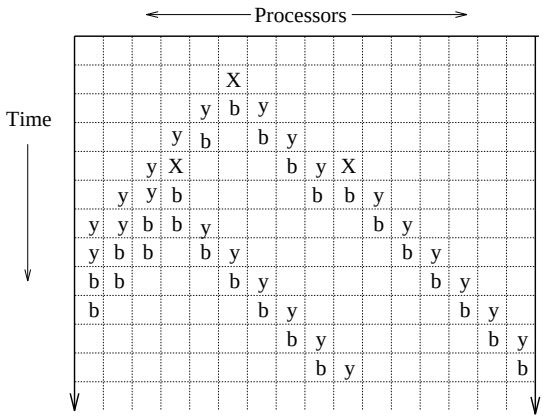

Figure 2: The figure depicts the operation of Σ when three errors have occurred. There is one pair of time-like errors.

A tree-code failure is not the only reason for the transmission of incorrect data. It may also be that one of the neighbors of p transmitted wrong data in the preceding round and consequently, p transmitted erroneous data. If such an event occurs – specifically, if there was no processor tree code error at (p,t) but nevertheless p transmitted erroneous data (which can only occur for the indicated reason), we say that p made a propagated error, and set $\textsc{Action}(p,t) = \mathsf{y}$. A number of such errors are indicated in figures 2 and 3.

Consider the position $(6,3)$ in figure 2. Since $\textsc{Action}(6,3)$ is not an x or bx, processor 6 decodes each incoming message correctly. Therefore it detects that the data it transmitted at the end of the last time step does not correspond to the preceding incoming data stream: thus, an inconsistency in its transcript. Hence processor 6 backs up one step. Now consider the position $(7,4)$. At this stage processor 7, too, observes an inconsistency in its transcript. The inconsistency arises because processor 7 has transmitted 3 messages of Π to its neighbors even though it has received one message from 6. Thus, processor 7's response string is not a prefix of the correct response string (which is of length 2), and therefore processor 7 now backs up one step. Both of these actions are denoted by b in figure 2. In general, we set $\textsc{Action}(p,t) = \mathsf{b}$ if (a) There is no tree code error at p at time t, and p backs up; (b) For at least one of p's out-neighbors q, there was a character in the outgoing transcript $w^{(p,q)}(t-1)$ which was incorrect. (I.e. the character did not agree with that which would have been transmitted in a noiseless run of Π.) The reader can now verify that figure 2 is a faithful representation of the execution of the protocol, for the given pattern of errors.

If condition (a) but not (b) is satisfied, we set $\textsc{Action}(p,t) =$ by. Consider figure 3. Here an erroneous backup, bx, occurs at position $(7,4)$, and causes a series of by backups. All of these backups are spurious, and erase the good simulation steps that occurred in the positions marked by a thick dot, •. In this example those steps are promptly re-simulated in

[4] The various strings and the relationships between them are described by the following diagram:

$$\begin{array}{ccc} x & \xrightarrow{\mathcal{T}\text{-coded channel}} & \hat{x} \\ \text{parse}\downarrow\uparrow\text{protocol} & & \downarrow\text{parse} \\ w & & \hat{w} \end{array}$$

The strings $w, \hat{w}$ are over $\{0,1\}$; x and $\hat{x}$ are over $\{0,1,\mathrm{bkp}\}$

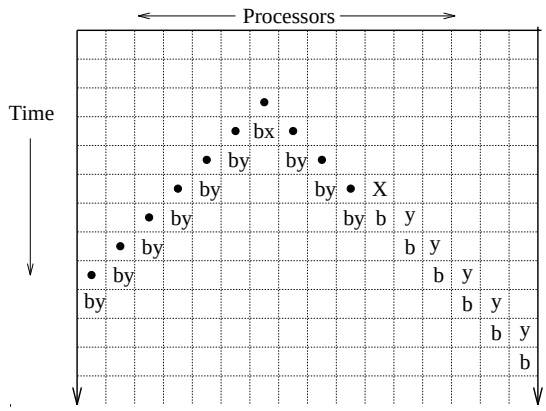

Figure 3: This figure shows an instance of a spurious backup.

the unmarked rounds following the spurious backups.

Observation 4.0.1 *If a processor does not make a tree code error (in which case* ACTION *is either* x *or* bx*), does not back up (in which case* ACTION *is either* b, by *or* bx*), and is not transmitting data based on erroneous information (in which case* ACTION *is* y*) then it is necessarily successfully simulating a round of* Π*. In that case we say that* ACTION = progress*.*

We leave the figures blank in such progress positions.

We now define the *real progress* $\mathrm{RP}(p,t)$, the key measure of the progress of our simulation. Unlike $\mathrm{AT}(p,t)$, it is not one that can be determined by the processors while they are running the protocol. The real progress $\mathrm{RP}(p,t)$ is simply the number of steps of the original protocol Π that have been correctly simulated by processor p at time t. Thus for example in figure 2, the real progress at position $(4,5)$ is 3. By way of contrast $\mathrm{AT}(4,5) = 5$. In figure 3, $\mathrm{RP}(5,5) = 5$, and $\mathrm{RP}(5,6)$ is 4. This decrease in the value of the real progress is due to the spurious back up which erased a step of meaningful computation.

A consequence of the definitions is that $\mathrm{RP}(p,t) \leq \mathrm{AT}(p,t)$.

5 Analysis

5.1 Overview

We will prove our result by showing that with high probability, the real progress at a processor at a given time t (measured in rounds during each of which a single tree code character is transmitted over each channel), is within a constant factor of t.

There are two key components to our analysis. The first is to associate the delay at processor p at time t with a specific, *time-like* sequence of processor tree code errors in the history cone of (p,t). We say that (p,t) and (p',t') are time-like if a signal leaving processor p at time t can reach processor p' by time t' (or vice versa): thus (p,t) and (p,t') are always time-like, and (p,t) and $(p,t+1)$ are time-like if p and p' are neighbors.

If (p,t) and (p',t') are not time-like we say they are *space-like*. A set $\{(p_1,t_1),...,(p_n,t_n)\}$ are time-like or space-like if every two of them are. The history cone at (p,t) consists of all time-like predecessors of (p,t), and (p,t) itself.

The second component of our analysis is to argue that with high probability, the number of processor tree code errors (henceforth abbreviated as $\mathcal{T}$-errors) on any single time-like path is small. Since $\mathcal{T}$-errors are not independent of each other, we accomplish this objective by associating with any large time-like sequence of $\mathcal{T}$-errors a large number of "nearly time-like" processor character errors. Since character errors are essentially independent, a Chernoff bound can then be established.

These two components are summarized in the following two lemmas:

Lemma 5.1.1 *If for a processor* p*,* $\mathrm{RP}(p,t) = t - \ell$*, then there is a time-like sequence of at least* $\ell/2$ $\mathcal{T}$*-errors in the time history cone of* p *at* t*.*

Lemma 5.1.2 *Choose the dilation* k *of the Shannon code* χ *to be* $c\log(d+1)/C$*, for a sufficiently large constant* c*. For any fixed time-like sequence* $\{(p_i,i) : 1 \leq i \leq t\}$*, the probability that there are more than* $\frac{t}{4}$ $\mathcal{T}$*-errors in the time-like sequence is less than* $\frac{1}{(2(d+1))^t}$*.*

Since the number of time-like sequences in a simulation of length t is at most $(d+1)^t N$ where d is the largest in-degree of any processor, the probability of any time-like sequence having more than $t/4$ errors is at most $(d+1)^t N/(2(d+1))^t = N2^{-t}$ and the main theorem 1.1 follows if Σ is run for twice as many rounds as Π.

The second component of the proof is accomplished by extending a method used in [26]. In this paper, most of our efforts will be devoted to the first component: associating delay with a time-like sequence of tree-code errors.

5.2 Simulation Delays and Tree Code Errors

In this section we link simulation delays and time-like sequences of tree code errors. Let $\mathrm{X}(p,t)$ denote the size of longest time-like sequence of x's and bx's in the history cone of (p,t). Our main task is to show:

Proposition 5.2.1 *For any processor* p *and any moment in time* τ*,*

$$\tau \leq \mathrm{RP}(p,\tau) + \mathrm{X}(p,\tau) + \mathrm{B}(p,\tau)$$

It will then be relatively straightforward to obtain lemma 5.1.1 from this proposition.

Proof: We prove the proposition by induction on time τ. We start with $\tau = 0$ in which case all the terms of the inequality are 0. For the induction we may assume that for time $t \leq \tau + 1$ and for any processor p in the network, $t \leq \mathrm{RP}(p,t) + \mathrm{X}(p,t) + \mathrm{B}(p,t)$. We now show

$$\tau + 1 \leq \mathrm{RP}(p,\tau+1) + \mathrm{X}(p,\tau+1) + \mathrm{B}(p,\tau+1).$$

The proof is by case analysis according to the classification $\textsc{Action}(p,\tau+1)$ described in section 4. There are four easy cases which we deal with first. If $\textsc{Action}(p,\tau+1) =$ progress, then $\mathrm{RP}(p,\tau+1) = \mathrm{RP}(p,\tau) + 1$ and the other quantities are unchanged from (p,τ) (except that X may increase); the induction follows. If $\textsc{Action}(p,\tau+1) =$ x then $\mathrm{X}(p,\tau+1) \geq \mathrm{X}(p,\tau)+1$ and again, the other quantities are unchanged from (p,τ) the induction follows. Similarly if $\textsc{Action}(p,\tau+1) =$ b then $\mathrm{B}(p,\tau+1) = \mathrm{B}(p,\tau) + 1$ and the other quantities are unchanged from (p,τ) except that

X might possibly increase; the induction follows. Finally if $\textsc{Action}(p, \tau+1) = \mathsf{bx}$ then both $\mathrm{X}(p,\tau)$ and $\mathrm{B}(p,\tau)$ increase by 1 while $\mathrm{RP}(p,\tau)$ may decrease. However $\mathrm{RP}(p,\tau)$ can decrease by at most 1 so the induction follows.

We now consider the two nontrivial cases, beginning with $\textsc{Action}(p,\tau+1) = \mathsf{y}$.

Lemma 5.2.1 *Let* $\textsc{Action}(p,t+1) = \mathsf{y}$. *Then the following two conditions hold.*

1. *For every processor* q *such that* $(q,p) \in \mathcal{N}$, $\mathrm{B}(q,t) \leq \mathrm{B}(p,t)$.
2. *There is a* q *such that* $(q,p) \in \mathcal{N}$ *and* $\mathrm{RP}(q,t) < \mathrm{RP}(p,t)$.

Consequently, there is a q *such that* $(q,p) \in \mathcal{N}$, *and*

$$\mathrm{RP}(q,t) + \mathrm{B}(q,t) < \mathrm{RP}(p,t) + \mathrm{B}(p,t)$$

Proof: Since $\textsc{Action}(p,t+1)$ is neither an x nor a bx, there was no tree code error at processor p at time $t+1$. Hence for each incoming edge $(q,p) \in \mathcal{N}$, $\hat{w}^{(q,p)} = w^{(q,p)}$. Therefore if (1) were violated, p would have backed up; while if (2) were violated, the data transmitted on each outgoing channel would have been correct. □

For the q provided by the lemma we have:

$$\begin{aligned}\mathrm{RP}(q,\tau) + \mathrm{B}(q,\tau) &< \mathrm{RP}(p,\tau) + \mathrm{B}(p,\tau) \\ &= \mathrm{RP}(p,\tau+1) + \mathrm{B}(p,\tau+1)\end{aligned}$$

Further since $(q,p) \in \mathcal{N}$, $\mathrm{X}(q,\tau) \leq \mathrm{X}(p,\tau+1)$. Consequently,

$$\begin{aligned}&\mathrm{RP}(q,\tau) + \mathrm{X}(q,\tau) + \mathrm{B}(q,\tau) \\ &\quad< \mathrm{RP}(p,\tau+1) + \mathrm{X}(p,\tau+1) + \mathrm{B}(p,\tau+1)\end{aligned}$$

Applying the induction hypothesis at q at time τ, we obtain:

$$\tau < \mathrm{RP}(p,\tau+1) + \mathrm{X}(p,\tau+1) + \mathrm{B}(p,\tau+1)$$

Which completes the induction in this case due to the strict inequality.

The final case is that $\textsc{Action}(p,\tau+1) = \mathsf{by}$. Here $\mathrm{B}(p,\tau)$ increases by 1 while $\mathrm{RP}(p,\tau)$ decreases by 1. Note that $\mathrm{X}(p,\tau)$ does not necessarily increase. However we can establish:

Lemma 5.2.2 *Let* $\textsc{Action}(p,t+1) = \mathsf{by}$. *Then there is a processor* q *such that* $(q,p) \in \mathcal{N}$ *and* $\mathrm{RP}(q,t) < \mathrm{RP}(p,t) - 1$.

Proof: The classification by implies that $\mathrm{RP}(p,t) = \mathrm{AT}(p,t)$; and that $\mathrm{AT}(q,t) < \mathrm{AT}(p,t)$. Because of the parity condition on AT it follows that $\mathrm{AT}(q,t) < \mathrm{AT}(p,t) - 1$, and since $\mathrm{RP}(q,t) \leq \mathrm{AT}(q,t)$, the lemma follows. □

Let q be as provided by the lemma, and consider two cases. In the first case suppose that $\mathrm{B}(q,\tau) \leq \mathrm{B}(p,\tau) + 1$. Then

$$\begin{aligned}&\mathrm{RP}(q,\tau) + \mathrm{X}(q,\tau) + \mathrm{B}(q,\tau) \\ &\quad< \mathrm{RP}(p,\tau) + \mathrm{X}(p,\tau+1) + \mathrm{B}(p,\tau)\end{aligned}$$

Since $\mathrm{RP}(p,\tau) + \mathrm{B}(p,\tau) = \mathrm{RP}(p,\tau+1) + \mathrm{B}(p,\tau+1)$, we find

$$\begin{aligned}&\mathrm{RP}(q,\tau) + \mathrm{X}(q,\tau) + \mathrm{B}(q,\tau) \\ &\quad< \mathrm{RP}(p,\tau+1) + \mathrm{X}(p,\tau+1) + \mathrm{B}(p,\tau+1)\end{aligned}$$

Applying the inductive hypothesis at q at time τ, we obtain:

$$\tau < \mathrm{RP}(p,\tau+1) + \mathrm{X}(p,\tau+1) + \mathrm{B}(p,\tau+1)$$

which completes the induction for the subcase $\mathrm{B}(q,\tau) \leq \mathrm{B}(p,\tau) + 1$.

In the second case $\mathrm{B}(q,\tau) > \mathrm{B}(p,\tau) + 1$. As observed in the proof of the lemma 5.2.2, $\mathrm{RP}(p,t) = \mathrm{AT}(p,t)$; therefore

$$\mathrm{RP}(p,\tau) = \tau - 2\mathrm{B}(p,\tau)$$

At (q,τ) we know only that

$$\mathrm{RP}(q,\tau) \leq \tau - 2\mathrm{B}(q,\tau)$$

Subtracting we have:

$$\mathrm{RP}(p,\tau) + 2\mathrm{B}(p,\tau) \geq \mathrm{RP}(q,\tau) + 2\mathrm{B}(q,\tau),$$

and applying the assumption $\mathrm{B}(q,\tau) > \mathrm{B}(p,\tau) + 1$,

$$\mathrm{RP}(p,\tau) + \mathrm{B}(p,\tau) \geq \mathrm{RP}(q,\tau) + \mathrm{B}(q,\tau) + 1.$$

As in the previous case we now use the fact that $\mathrm{RP}(p,\tau) + \mathrm{B}(p,\tau) = \mathrm{RP}(p,\tau+1) + \mathrm{B}(p,\tau+1)$, and find:

$$\begin{aligned}&\mathrm{RP}(q,\tau) + \mathrm{X}(q,\tau) + \mathrm{B}(q,\tau) \\ &\quad< \mathrm{RP}(p,\tau+1) + \mathrm{X}(p,\tau+1) + \mathrm{B}(p,\tau+1)\end{aligned}$$

Which, applying the inductive hypothesis at q at time τ, settles this last case of the induction for the proposition. □

In order to obtain lemma 5.1.1 we observe:

Corollary 5.2.3 *For any processor* (p,t),

$$\mathrm{RP}(p,t) \geq t - 2\mathrm{X}(p,t)$$

Proof: Using the proposition to substitute for X we have:

$$\begin{aligned}&\mathrm{RP}(p,t) - t + 2\mathrm{X}(p,t) \\ &\quad\geq t - \mathrm{RP}(p,t) - 2\mathrm{B}(p,t)\end{aligned}$$

As already noted, the latter quantity is always nonnegative. □

The corollary implies lemma 5.1.1, the first of the two main results needed for theorem 1.1, because $\ell/2 = (t - \mathrm{RP}(p,t))/2 \leq \mathrm{X}(p,t)$.

5.3 Example

Figure 4 shows a large simulation and, by way of example, a *critical path* of tree code errors, which justifies the delays that have occurred in the protocol. Observe that in the four easy cases, the critical path leads upward, to the same processor at the previous moment; while in the two cases y and by the critical path leads up to a neighboring processor at the previous moment.

5.4 Tree Code Errors and Character Errors

In this section we sketch the proof of the second main lemma, 5.1.2. The difficulty in establishing this lemma is that $\mathcal{T}$-errors are not independent. Our proof is an extension of a method used in [26].

Proof of lemma 5.1.2. We begin by taking the union bound over the $\binom{t}{t/4} < 2^t$ different ways in which $t/4$ tree code errors could appear on this time-like path. Now fix an error pattern on the path. We need the following observation:

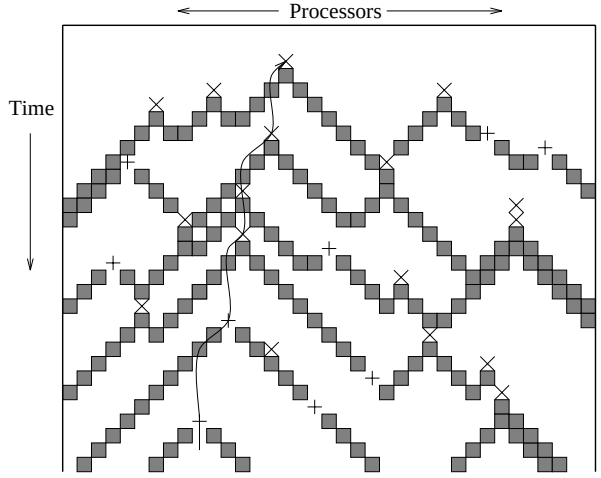

Figure 4: A time-like sequence of 6 $\mathcal{T}$-errors that account for the 12 timesteps of lost computation. Errors of type x are indicated by $\times$, errors of type bx are indicated by $+$, and the shaded boxes indicate backup steps of type b or by.

Observation 5.4.1 *By suitable choice of the constant c in lemma 5.1.2, the probability of occurrence of any set of n processor character errors can be bounded by* $\exp(-c'n)$ *for arbitrarily large c'.*

Proof: Although, for reasons noted earlier, character errors are not exactly independent, nevertheless the conditional probability of any character error, given any information about the occurrence of other character errors, is bounded by $d\exp(-Ck) \leq \exp(-c')$. □

Fix any processor p, and suppose the fixed error pattern assigns l specific $\mathcal{T}$-errors to that processor. We will show that the probability of this occurring is exponentially small in l; moreover, by lemma 5.4.1, this will be true conditional on any events at the other processors. The lemma 5.1.2 will follow.

With reference to the argument in [26]: The l tree code errors must be contained within the union of the "error intervals" at p. There is a disjoint collection of error intervals covering at least $l/2$ of the tree code errors. If the union of these disjoint intervals is of length m, the probability of their occurrence is exponentially small in m. If m is substantially larger than $l/2$ there are many ways to select such a set of intervals, but this union bound is dominated by the $\exp(-\Omega(m))$ probability of occurrence of any particular set of intervals. □

6 Data Broadcast on a Chain of Processors

The protocol described above is based on tree codes. As mentioned earlier, significant difficulties remain concerning the computational feasibility of this protocol.

In this section we provide a computationally efficient protocol for a special case of the problem. We study the simplest of network topologies, the chain, and the simplest of network tasks, the broadcast of data. Let the first processor in a chain of N processors have B bits to be communicated to all the other processors. In a noiseless setting, by pipelining, this task can be carried out in time $N + B - 2$.

The naive strategy of sending the block in error correcting code would take $\Theta(B \cdot N)$ time – since each intermediate processor needs to receive the entire codeword before it can proceed any further. One could divide the block of size B into smaller blocks of size b. The running time would then reduce to $O(N \cdot b + B)$. This reduction in running time, however, comes at the cost of increasing the failure probability from roughly $e^{-\Omega(B)}$ to $e^{-\Omega(b)}$. Moreover even if we settle for a constant failure probability, the multiplicative time overhead cannot be reduced below $\log N$ in this way.

6.1 Protocol

In this section we describe a simple protocol that achieves speed without compromising the reliability of execution. The basic idea here is to encode the message block in error correcting code; then divide the codeword into small chunks, and send each chunk to the end of the chain by successively transmitting it across subchains of the chain, each time using the protocol recursively. Finally after all the chunks have reached the last processor, they are combined and the message is obtained by using the decoding algorithm for the error correcting code.

Observe that the computation in this method is dominated by the final decoding of the error-correcting code, and thus the method is computationally essentially as efficient as any two-processor data transmission.

Let $M = \max\{B, N\}$. Without loss of generality the number of processors and the message length are both M. The protocol P_M is described by the following four steps.

1. Encode the M bits in a 25% error correcting code $\mathcal{E}$. This results in a constant factor blow-up in the message size. Let $\alpha > 4$ be such that $|\mathcal{E}(M)| = \alpha M$.

2. Partition $\mathcal{E}(M)$ into chunks of bits, each of size $m = \log^2 M$. There are $\alpha M/m$ such chunks.

3. Use protocol P_m to transfer each chunk of m bits across a subchain of length m. At the end of the subchain, use P_m again to send the chunk to the end of the next subchain of length m; and so on until the end of the chain. Pipeline the chunks one behind the other.

4. Finally when every chunk has arrived at the end of the chain, invert the error-correcting code using the chunks of data provided by the recursive protocols.

6.2 Analysis

Theorem 6.1 *A B bit message can be transmitted across an array of N processors interconnected by binary symmetric channels in time $O((B+N)\cdot e^{O(\log^*(B+N))})$. The probability of an error occurring in transmission is at most $e^{-\Omega(N+B)}$.*

Proof: The base case for the analysis, constant M, is trivial and requires only the appropriate selection of constants. In the remainder of the proof we assume that M is large enough for asymptotic inequalities to hold.

There are two issues to be resolved: the running time and the error probability. We consider the running time first:

The total running time has two components. The first is the *processor broadcast time*, b_M. This is the length of time each processor is occupied in broadcasting bits. (Thus in pipelining of B bits in a noiseless chain this would be B.) The second is *pipeline latency*, l_M. This is the delay between

the moment the first bit is transmitted at the first processor, and when the first bit is received at the last processor. (Thus in pipelining in a noiseless chain of length N this would be $N-1$.) The runtime of our protocol is $b_M + l_M$.

It is convenient now to define the function $L^*(M)$ to be the number of applications of the function $\log^2$ necessary before M is brought below some constant threshold. Observe that $L^*(M)$ is within a constant factor of $\log^*(M)$.

We show by induction that $b_M \leq M\alpha^{L^*M}$; and (for any $\beta > 0$), $l_M \leq M(\alpha+\beta)^{L^*M}$. For the first of these, observe that $b_M = b_m \alpha M/m = m\alpha^{L^*M-1}\alpha M/m = M\alpha^{L^*M}$. For the second, observe that

$$l_M = (b_m + l_m)M/m - l_m < (b_m + l_m)M/m$$

$$= (m\alpha^{L^*m} + m(\alpha+\beta)^{L^*m})M/m$$

$$= M(\alpha^{L^*M-1} + (\alpha+\beta)^{L^*M-1}) < M(\alpha+\beta)^{L^*M}.$$

We next show by induction that the error probability q_M is bounded by $exp(-M)$. By induction the probability of any of the recursive protocols erring is $exp(-m)$, and by taking the union bound over the number of subchains, we can bound the probability of a chunk arriving incorrectly at the last processor by $exp(-m)M/m$. The protocol P_M can err only if at least a quarter of the $\alpha M/m$ chunks arrive incorrectly at the last processor, so by the Chernoff bound,

$$q_M \leq [(4q_m)^{1/4}(4(1-q_m)/3)^{3/4}]^{\alpha M/m}$$

$$\leq q_m^{\alpha M/4m}(4(1/3)^{3/4})^{\alpha M/m}$$

$$\leq e^{-\alpha M/4}(4(1/3)^{3/4})^{\alpha M/m} < e^{-M}.$$

□

7 Lower bound

We demonstrate the following lower bound on the problem of broadcasting one bit from start to end of a chain of length n, in which each link is a binary symmetric channel, erring with probability $(1-\delta)/2$ on each transmission:

Theorem 7.1 *Suppose that each input bit 0,1 is equally likely. Then for any $\gamma < 1$, for sufficiently large values of n, any bit computed at processor n at time $(n-1)\gamma/\delta + 1$ is equal to the input bit with probability at most $1/2 + o(1)$.*

The argument is through information theory. We will show that the information available at processor n regarding the input value, at the time described above, tends to 0 in n. The implication for error probability follows by standard arguments.

We begin with a lemma regarding information. This is a slightly modified version of a lemma of Pippenger [18].

Let X be random variable which we interpret as "data". Let Y be a random variable which depends on X through some arbitrary channel. We will think of Y as referring to the entire sequence of receptions at some processor k of the chain, up through time t. Let $Z = (Z_1, Z_2)$ be a random variable which depends on Y through a binary symmetric channel which errs with probability $(1-\delta)/2$. We think of Z as the entire sequence of receptions at processor $k+1$ of the chain, up through time $t+1$. Z_2 denotes just the last bit received over the channel, at time $t+1$, while Z_1 denotes the entire sequence of previous receptions across the channel.

Lemma 7.0.1 $I(X;Z) \leq (1-\delta)I(X;Z_1) + \delta I(X;Y)$.

Proof: Let H be a binary random variable which is 1 w. prob. $1-\delta$. Let R be a binary random variable which is 1 w. prob. $1/2$. Z_1 is equivalent to a binary variable defined as follows: If $H = 1$ then it is equal to R; otherwise it is equal to the bit sent by Y.

In the former case $I(X;Z|H=1) = I(X;Z_1)$; in the latter case we use just the data processing lemma, to obtain $I(X;Z|H=0) \leq I(X;Y)$.

$I(X;Z) \leq I(X;ZH) \leq (1-\delta)I(X;Z_1) + \delta I(X;Y)$. □

The inequality obtained above implies that the information at processor k at time t can be bounded by the function $f(k,t)$ defined in the following way:

- $f(0,t) = 1 \forall t$.
- $f(k,t+1) = (1-\delta)f(k,t) + \delta f(k-1,t)$.

This function is:

$$f(k,t) = \delta \sum_{j=1}^{t-(k-1)} \delta^{k-1}(1-\delta)^{t-j-(k-1)} \binom{t-j}{k-1}.$$

Each term in this summation is the probability that, in a Bernoulli process where Heads come up with probability δ, exactly $k-1$ Heads came up in the first $t-j$ trials. For $t - j \leq (k-1)/\delta$, it is easily shown that the largest term in the summation is that for which $j = 1$. Hence

$$f(k,t) \leq (t-k+1)\delta^k(1-\delta)^{t-k}\binom{t-1}{k-1}.$$

Now parameterize $t(k) = (k-1)\gamma/\delta + 1$.

Corollary 7.0.2 *For any $\gamma < 1$, $\lim_{k\to\infty} f(k,t(k)) = 0$.*

Proof: The Chernoff bound shows that

$$\delta^{k-1}(1-\delta)^{t(k)-k}\binom{t(k)-1}{k-1} \leq e^{-2(k-1)\delta(1-\gamma)^2/\gamma}.$$

Thus

$$f(k,t) \leq (k-1)(\gamma-\delta)e^{-2(k-1)\delta(1-\gamma)^2/\gamma}.$$

This converges to 0 in the limit $k \to \infty$. □

8 Discussion

The question of explicitly constructing tree codes is a beautiful problem which remains unresolved. This question may be formalized by asking for an algorithm which in time $poly(n)$ names the letter of S on any specified edge of a tree of depth n.

In addition, it would be highly desirable to make the computations associated with the protocol for theorem 1.1, effective. The max-likelihood criterion we employ for decoding tree codes requires a computational overhead that is exponential in the communication complexity.

References

[1] R. Aleliunas. Randomized parallel communication. In *Proceedings of the Symposium on the Principles of Distributed Systems.*, 1982, pages 60–72.

[2] T. M. Cover and J. A. Thomas. *Elements of Information Theory.* Wiley, 1991.

[3] 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.

[4] 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.

[5] 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.

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

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

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

[9] 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.

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

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

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

[13] M. Luby. On the parallel complexity of symmetric connection networks. *U. Toronto CS Dept. Tech. Report # 214*, 1988.

[14] C. Martel, R. Subramonian, and A. Park. Asynchronous PRAMs are (almost) as good as Synchronous PRAMs. In *Proceedings of the 31st Annual Symposium on Foundations of Computer Science*, pages 590–599, 1990.

[15] N. Nisan Personal Communication.

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

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

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

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

[20] N. Pippenger, G. D. Stamoulis, and J. N. Tsitsiklis. On a lower bound for the redundancy of reliable networks with noisy gates. *IEEE Transactions on Information Theory*, 37(3):639–643, 1991.

[21] A.G. Ranade. A Simpler Analysis of the Karp-Zhang Parallel Branch-and-Bound Method. *University of California, TR, UCB/CSD 90/586*, 1990.

[22] A.G. Ranade. How to emulate shared memory. *JCSS*, 42(3):307-326, June, 1991.

[23] 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.

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

[25] 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.

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

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

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

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

[30] A. L. Toom. Nonergodic multidimensional systems of automata. *Problems Inform. Transmission*, 10:239–246, 1974.

[31] A. L. Toom. Stable and attractive trajectories in multicomponent systems. In R. L. Dobrushin, editor, *Multicomponent Systems*, volume 6 of *Adv. Probab.*, pages 549–575. Dekker, 1980.

[32] A. L. Toom. Estimates for the measures describing the behavior of stochastic systems with local interaction. In Kryukov Dobrushin and Toom, editors, *Interactive Markov Processes and Their Application to the Mathematical Modelling of Biological Systems.* Acad. Sci. USSR, 1982.

[33] B. S. Tsirel'son. Reliable information storage in a system of locally interacting unreliable elements. In *Interacting Markov Processes in Biology*, volume 653 of *Lecture Notes in Mathematics.* Springer-Verlag, 1978.

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

[35] 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.

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