DETERMINISTIC SIMULATION OF RANDOMIZED PROTOCOLS OVER ARBITRARY NETWORKS OF NOISY CHANNELS

by

S. Venkatesan and V. Anantharam

Memorandum No. UCB/ERL M96/65

30 August 1996

DETERMINISTIC SIMULATION OF RANDOMIZED PROTOCOLS OVER ARBITRARY NETWORKS OF NOISY CHANNELS

by

S. Venkatesan and V. Anantharam

Memorandum No. UCB/ERL M96/65

30 August 1996

ELECTRONICS RESEARCH LABORATORY

College of Engineering
University of California, Berkeley
94720

Deterministic simulation of randomized protocols over arbitrary networks of noisy channels*

S. Venkatesan† V. Anantharam‡§

Abstract

Suppose the input to a function is split between several processors connected by a network of binary channels, and the processors know an interactive protocol by which they can compute the function in N steps of communication, provided the channels are all noiseless. Since all practical channels are noisy, it is important to study the effect of channel noise on the complexity and reliability of the protocol. In this direction, Rajagopalan and Schulman recently proved that any N-step noiseless-network protocol can be simulated over a network of identical and independent binary symmetric channels (with the same topology) in $O((N/C)\log d)$ steps, while incurring a simulation failure probability of $2^{-\Omega(N)}$. Here, d is the maximum in-degree of any processor in the network, and C is the capacity of each channel. We show that this result can be strengthened in the following way: any N-step private-coin *randomized* protocol that computes a function correctly with probability at least $1-\epsilon$ (in the noiseless case) can be *deterministically* simulated over the noisy network in $O\left(\frac{Nm}{1-C}\right) + O\left(\frac{N\log d}{C}\right)$ steps, while incurring an error probability in computing the function of at most $\epsilon + 2^{-\Omega(N)}$. Here, m is the number of channels in the network.

*Research supported by NSF IRI 9005849, IRI 9310670, NCR 9422513, and the AT&T Foundation.

†Cornell University and U.C. Berkeley.

‡Univ. of California, Berkeley.

§Address all correspondence to the second author: 570 Cory Hall, Dept. of EECS, U.C. Berkeley, Berkeley, CA 94720.

1 Introduction

Consider the following general model for distributed computation: the input to a function is split between several processors connected by a network of binary channels, and all the processors must compute the function by exchanging information over the network, according to a synchronous interactive protocol. This protocol allows them to perform the computation on any input with a certain number of steps of communication, provided the channels in the network are all noiseless.

In this model, it is important to study the effect of channel noise on the complexity (number of steps of communication) and reliability (error probability in computing the function) of the protocol, since all practical channels are noisy. This problem was first addressed by L. Schulman in [2] for the special case of two processors connected to each other by independent binary symmetric channels (BSCs) with the same crossover probability of δ ($0 < \delta < 1/2$). He proved that any N-step noiseless-channel interactive protocol could be simulated over the BSCs in $O(N/C)$ steps ($C = 1 + \delta \log \delta + (1-\delta)\log(1-\delta)$ is the capacity of each channel), while incurring a simulation failure probability of $2^{-\Omega(N)}$. (Here, a simulation failure is said to occur if at least one of the processors does not arrive at the same computed value of the function in the simulation as in the original protocol.)

This result is analogous to Shannon's coding theorem for (one-way) data transmission across a noisy channel, in that it establishes the possibility of simulating long enough protocols over noisy channels with a constant factor overhead in the number of transmissions, while incurring an arbitrarily small failure probability. Note, however, that Shannon's theorem cannot be directly applied here because of the interactive nature of the communication; in general, neither processor knows all its transmissions ahead of time, and therefore cannot code large blocks of data as in the data transmission case. The key idea in [2] is the use of *tree codes* as a mechanism for recovering from errors caused by channel noise.

Subsequently, in [1], Rajagopalan and Schulman extended this result to the case of an arbitrary network of processors, in which each channel is an independent BSC with crossover probability δ ($0 < \delta < 1/2$). They proved that any N-step noiseless-network protocol could be simulated over the noisy network in $O((N/C)\log d)$ steps, while incurring a simulation failure probability of $2^{-\Omega(N)}$. Here, d is the maximum in-degree of any processor in the network.

In both results, if the original protocol happens to be an N-step private-

coin randomized protocol which computes a function correctly on any input with probability at least $1-\epsilon$, then a) the simulation requires the same randomness resources at each processor as the original protocol does, and b) the probability that at least one of the processors fails to compute the function correctly in the simulation is upper bounded by $\epsilon + 2^{-\Omega(N)}$.

However, this result can be strengthened in the following sense: the processors can actually turn the channel noise to their advantage by using it to generate all the randomness they need in the protocol, so that, in the simulation, all external sources of randomness can be done away with. This is the subject of the present paper. The price to be paid for the deterministic simulation is an increase in the number of steps and in the error probability. We prove that any N-step private-coin randomized protocol that computes a function correctly with probability at least $1-\epsilon$ can be deterministically simulated over the noisy network in

$$O\left(\frac{Nm}{h(\delta)}\right) + O\left(\frac{N \log d}{1-h(\delta)}\right)$$

steps, while incurring an error probability in computing the function of at most $\epsilon + 2^{-\Omega(N)}$. Here, m is the number of channels in the network, d is the maximum in-degree of any processor, and $h(\delta) = -\delta \log \delta - (1-\delta)\log(1-\delta)$. This extends a similar result proved in [3] for the two-processor case.

The problem and result are stated more precisely in the following section, after some preliminary definitions.

2 Preliminaries

We will assume that the network of processors is represented by a directed graph $G = (V, E)$. Here, V is the (finite) set of processors, and $E \subseteq V \times V$ describes their interconnections. $(q, r) \in E$ means there is a binary channel *from q to r*. For each $q \in V$, let $V_{in}(q) = \{p : (p, q) \in E\}$, $d_{in}(q) = |V_{in}(q)|$, $V_{out}(q) = \{r : (q, r) \in E\}$, and $d_{out}(q) = |V_{out}(q)|$.

Without much loss of generality, we may assume that the inputs of all the processors are drawn from a common finite set $\mathcal{X}$. Let $\mathbf{x}$ represent the vector of all their inputs, and let $f(\mathbf{x})$ be the common function that they wish to compute. Let $\mathcal{Y}$ be the finite set in which f takes values.

As mentioned before, the processors are assumed to know a private-coin randomized protocol that helps them compute f with a certain error probability, provided the network is noiseless. We will now describe this protocol (call it Π) more carefully.

Π runs in N *steps*, indexed $n = 1, 2, \ldots, N$. In step n, processor q does the following in sequence and in synchronism with all other processors:

- It randomly chooses the $d_{out}(q)$-tuple of bits to transmit on its out-links in step n, with a distribution that depends only on its own input and the $d_{in}(q)$-tuple of bits that it received on its in-links in each of the previous $n-1$ steps.
- It then transmits these bits on its out-links.
- Finally, it receives the bits sent on its in-links in step n.

At the end of step N, q randomly chooses an element of $\mathcal{Y}$, with a distribution depending only on its own input and the bits that it received on its in-links in the N steps, and takes the chosen element to be the value of $f(\mathbf{x})$. It should be mentioned here that all the random experiments performed by a processor are independent of each other and of the random experiments performed by all other processors.

The protocol guarantees that, for any input vector $\mathbf{x}$, the probability that all the processors compute $f(\mathbf{x})$ correctly is at least $1 - \epsilon$ (for some $0 \leq \epsilon < 1$). ϵ is called the error probability of Π.

Our objective is to simulate Π *deterministically* over a network described by the same graph G, in which each link is a binary symmetric channel of crossover probability δ $(0 < \delta < 1/2)$, and the channels operate independently.

If the simulation is to be deterministic, the randomness required by the processors must be generated by a deterministic procedure from the *noise* in the network (which is the only source of randomness). Now, as such, the distributions with which the processors make their random choices in Π are arbitrary. It will therefore be convenient to first modify Π to a new randomized protocol Π' which also runs in N steps, but in which each processor initially draws an element from a set of size K with a *uniform* distribution, and thereafter makes all its choices as deterministic functions of its input and random element (the processors draw their random elements independently). The price to be paid for this "uniformization" is a possible increase in the error probability, which depends on how large K is. In Section 3, we show that this increase can be made smaller than $(|V|/K) \cdot 2^{N|E|}$.

We will actually simulate Π', and not Π, over the noisy network. The uniform distributions required in Π' can be generated deterministically from

channel noise in t steps of communication, but for a failure probability that depends on how large t is. In Section 4, we show that this failure probability is at most

$$|V| \cdot \min_{0 \leq s \leq \delta} \left\{ tK \cdot 2^{-th(s)} + 2 \cdot 2^{-tD(s\|\delta)} \right\}. \tag{1}$$

Now, if $t \geq (1+\beta)N|E|/h(\delta)$ for some $\beta > 0$, then we can choose K so that

$$2^{N|E|} \cdot \frac{|V|}{K} + \min_{0 \leq s \leq \delta} \left\{ tK \cdot 2^{-th(s)} + 2 \cdot 2^{-tD(s\|\delta)} \right\} \leq 2^{-Nc(\beta,\delta)+o(N)} \tag{2}$$

for some $c(\beta, \delta) > 0$. It follows that by increasing the number of steps in the simulation by $(1+\beta)N|E|/h(\delta)$ and tolerating an increase of $|V| \cdot 2^{-Nc(\beta,\delta)}$ in the error probability with which f is computed, we can do away with the randomness resources required in Π.

The actual simulation of Π' on the noisy network is described in Section 5. This is more or less identical to the simulation outlined in [1]. However, there is an error in [1] in the definition of what constitutes a successful simulation. Setting this right requires some modifications to the original simulation, which are described here. The result is the same as in [1], viz., that once the random variables have been successfully generated, Π' can be simulated on the noisy network in $2kN$ steps, for any $k > \left(\frac{K_1}{1-h(\delta)}\right) \log(d+1)$, while incurring a simulation failure probability of at most

$$|V| \cdot 2^{-K_2(kC - K_1 \log(d+1))N}. \tag{3}$$

Here, $d = \max_{q \in V} d_{in}(q)$ and K_1 and K_2 are universal constants.

3 Uniformization of Π

Theorem 3.1 *For any integer $K \geq 1$, Π can be modified to another N-step randomized protocol Π', in which each processor only requires an independent random variable that is uniformly distributed over a set of size K. The error probability of Π' is at most $\epsilon + (|V|/K) \cdot 2^{N|E|}$.*

Proof: The idea of the proof is to approximate all relevant probabilities in Π by rational numbers with denominator K. In order to do this, it will be convenient to take the unit interval $[0, 1)$ with uniform measure as the sample space on which all the independent random experiments performed by each processor in Π are defined. We will now describe this construction

from the perspective of processor q. Let $x \in \mathcal{X}$ denote a possible input at q. In all that follows, the term interval will always mean a set of the form $[\alpha, \beta)$, where $0 \leq \alpha \leq \beta \leq 1$. If $\alpha = \beta$, the interval will be assumed to be empty.

To begin with, divide $[0,1)$ into $2^{d_{out}(q)}$ disjoint intervals,

$$\left\{I(a_1|x) : a_1 \in \{0,1\}^{d_{out}(q)}\right\},$$

with the length of $I(a_1|x)$ being proportional to the probability that, on input x, q transmits a_1 in step 1 of Π.

Then, for each $b_1 \in \{0,1\}^{d_{in}(q)}$, subdivide each $I(a_1|x)$ into $2^{d_{out}(q)}$ disjoint intervals,

$$\left\{I(a_1, a_2|x, b_1) : a_2 \in \{0,1\}^{d_{out}(q)}\right\},$$

with the length of $I(a_1, a_2|x, b_1)$ being proportional to the probability that, on input x, q transmits a_2 in step 2 of Π if it received b_1 in step 1.

In general, suppose $1 \leq n < N$, and $I(a_1^n|x, b_1^{n-1})$ has been defined for all $a_1^n \in \left(\{0,1\}^{d_{out}(q)}\right)^n$, and all $b_1^{n-1} \in \left(\{0,1\}^{d_{in}(q)}\right)^{n-1}$. Then, for each $b_n \in \{0,1\}^{d_{in}(q)}$, subdivide each $I(a_1^n|x, b_1^{n-1})$ into $2^{d_{out}(q)}$ disjoint intervals,

$$\left\{I(a_1^{n+1}|x, b_1^n) : a_{n+1} \in \{0,1\}^{d_{out}(q)}\right\},$$

with the length of $I(a_1^{n+1}|x, b_1^n)$ being proportional to the probability that, on input x, q transmits a_{n+1} in step $n+1$ of Π if it received b_1^n in the previous n steps.

Finally, for each $b_N \in \{0,1\}^{d_{in}(q)}$, subdivide each $I(a_1^N|x, b_1^{N-1})$ into $|\mathcal{Y}|$ disjoint intervals,

$$\left\{I(a_1^N, y|x, b_1^N) : y \in \mathcal{Y}\right\},$$

with the length of $I(a_1^N, y|x, b_1^N)$ being proportional to the probability that, after step N, q chooses y as the value of the function f, given that its own input is x and it received b_1^N in the N steps.

From this description, it should be obvious how q can make its random decisions in each step of Π with the correct probabilities, if it initially knows a random point drawn uniformly from $[0,1)$. For example, in step 1, q checks which interval of the form $I(a_1|x)$ contains its random point, and transmits the corresponding a_1. Suppose it receives b_1 in step 1. Then in step 2, it checks which interval of the form $I(a_1, a_2|x, b_1)$ contains its random point, and transmits the corresponding a_2, etc. Suppose q transmitted a_1^N and

received b_1^N in the N steps. Then, after step N, it checks which interval of the form $I(a_1^N, y|x, b_1^N)$ contains its random point, and takes the corresponding y to be the value of f.

Now, for each interval $I = [\alpha, \beta)$ occuring above, approximate α and β by the nearest rational number of the form m/K, $m = 0, 1, \ldots, K$, and denote the interval thus obtained by I'. Clearly, the length of I' is of the form m/K, and differs from the length of I by no more than $1/K$. Note that if the length of I is smaller than $1/K$ then I' is empty.

Suppose now that q initially knows a random element drawn uniformly from $\{0, 1/K, 2/K, \ldots, (K-1)/K\}$. Then, in the protocol Π', q proceeds as described above, except that it now checks the "primed" intervals in each step. Thus, in step 1, q transmits the a_1 such that $I'(a_1|x)$ contains its random element, etc. The protocol Π' is now well-defined. It remains to bound its error probability.

Let $\mathbf{x} = (x_q)_{q \in V}$ be any input vector. Let $b_n^q \in \{0,1\}^{d_{in}(q)}$ be the $d_{in}(q)$-tuple of bits that processor q receives in step n ($1 \leq n \leq N$), during the execution of Π or Π' on this input vector (these are of course random, and their distributions depend on which protocol is being executed). Let $\mathbf{b}^q = (b_1^q, \ldots, b_N^q)$, and $\mathbf{b} = (\mathbf{b}^q)_{q \in V}$. $\mathbf{b}$ can take on $2^{N|E|}$ values. Clearly, if $\mathbf{b}$ is known, we can figure out the transmissions of each processor in each step. Let $g_q(\mathbf{b}) \in \left(\{0,1\}^{d_{out}(q)}\right)^N$ be the sequence of q's transmissions in the N steps.

Now, let $f(\mathbf{x}) = y$. Let $\lambda(\mathbf{x})$ (resp. $\lambda'(\mathbf{x})$) be the probability in Π (resp. Π') that all the processors compute $f(\mathbf{x})$ correctly. Then,

$$\begin{aligned} \lambda(\mathbf{x}) &= \sum_{\mathbf{b}} \prod_{q \in V} |I(g_q(\mathbf{b}), y|x_q, \mathbf{b}^q)|, \\ \lambda'(\mathbf{x}) &= \sum_{\mathbf{b}} \prod_{q \in V} |I'(g_q(\mathbf{b}), y|x_q, \mathbf{b}^q)|. \end{aligned}$$

Here, $|I|$ denotes the length of the interval I. Therefore,

$$\begin{aligned} |\lambda'(\mathbf{x}) - \lambda(\mathbf{x})| &\leq \sum_{\mathbf{b}} \left| \prod_{q \in V} |I'(g_q(\mathbf{b}), y|x_q, \mathbf{b}^q)| - \prod_{q \in V} |I(g_q(\mathbf{b}), y|x_q, \mathbf{b}^q)| \right| \\ &\leq \sum_{\mathbf{b}} \sum_{q \in V} \left| |I'(g_q(\mathbf{b}), y|x_q, \mathbf{b}^q)| - |I(g_q(\mathbf{b}), y|x_q, \mathbf{b}^q)| \right| \\ &\leq \sum_{\mathbf{b}} \sum_{q \in V} \frac{1}{K} \end{aligned}$$

$$= 2^{N|E|} \cdot \frac{|V|}{K}.$$

Thus,

$$\begin{aligned} 1 - \lambda'(\mathbf{x}) &\leq 1 - \lambda(\mathbf{x}) + 2^{N|E|} \cdot \frac{|V|}{K} \\ &\leq \epsilon + 2^{N|E|} \cdot \frac{|V|}{K}. \end{aligned}$$

This completes the proof. □

4 Generating randomness from channel noise

The idea for generating randomness from channel noise is the following: suppose each processor transmits 0's on all its out-links for t steps. Then, processor q receives an i.i.d. 0-1 sequence of length $td_{in}(q)$, in which each bit is 1 with probability δ. Moreover, these sequences are independent from processor to processor. We may assume that $d_{in}(q) \geq 1$ for all q, so that each processor has at least t i.i.d. bits. Such a sequence of length t can be processed deterministically to generate a random variable that is uniformly distributed over a set of size K, except for a certain failure probability. This probability approaches zero exponentially as t increases, provided the "rate" $(\log K)/t$ is maintained at some fixed level below $h(\delta)$.

Theorem 4.1 *Let $Z_1, Z_2, \ldots, Z_t$ be i.i.d. 0-1 valued random variables, each of which is 1 with probability δ $(0 < \delta \leq 1/2)$. Then, there exist pairwise disjoint subsets $G_1, G_2, \ldots, G_K$ of $\{0,1\}^t$ such that*

$$Pr\{(Z_1, Z_2, \ldots, Z_t) \in G_k\} = \frac{1-\tau}{K}, \quad \text{for all } k \in \{1, 2, \ldots, K\}.$$

Here, $\tau = Pr\{(Z_1, Z_2, \ldots, Z_t) \notin \bigcup_k G_k\}$ is the probability of failure to generate randomness, and is at most

$$\min_{0 \leq s \leq \delta} \left\{ tK \cdot 2^{-th(s)} + 2 \cdot 2^{-tD(s\|\delta)} \right\}. \tag{4}$$

Thus, in t steps, all the processors can generate independent random variables uniformly distributed over a set of size K, but for an event of probability at most $|V|$ times the expression in (4).

Proof: Choose s to attain the minimum in (4). For $w \in \{0, 1, \ldots, t\}$, let T_w be the set of 0-1 sequences of length t which have exactly w 1's. Obviously, $|T_w| = \binom{t}{w}$. For each w satisfying $h(w/t) \geq h(s)$, let $G_1^w, \ldots, G_K^w$ be pairwise disjoint subsets of T_w, each of size exactly $\left\lfloor \binom{t}{w}/K \right\rfloor$ (the subsets are otherwise arbitrary). Let

$$G_k = \bigcup_{h(w/t) \geq h(s)} G_k^w, \qquad k = 1, 2, \ldots, K.$$

Clearly, $Pr\{(Z_1, Z_2, \ldots, Z_t) \in G_k\}$ is then the same for all k, since all the G_k's have the same number of sequences of any given type. It only remains to upper bound $\tau = Pr\{(Z_1, Z_2, \ldots, Z_t) \notin \bigcup_k G_k\}$. Note that

$$\begin{aligned} \tau &= \sum_{h(w/t) \geq h(s)} [|T_w| \bmod K] \delta^w (1-\delta)^{t-w} + \sum_{h(w/t) < h(s)} |T_w| \delta^w (1-\delta)^{t-w} \\ &\leq \sum_{h(w/t) \geq h(s)} K \cdot \delta^w (1-\delta)^{t-w} + \sum_{h(w/t) < h(s)} \binom{t}{w} \delta^w (1-\delta)^{t-w}. \end{aligned} \tag{5}$$

The first term in (5) equals

$$\begin{aligned} &\sum_{s \leq w/t \leq 1-s} K \cdot 2^{-t[h(w/t) + D(w/t \| \delta)]} \\ &\leq K \cdot 2^{-th(s)} \sum_{s \leq w/t \leq 1-s} 2^{-tD(w/t \| \delta)} \\ &\leq tK \cdot 2^{-th(s)}. \end{aligned}$$

By the Chernoff bound for the tails of a binomial distribution, the second term in (5) is at most $2^{-tD(s\|\delta)} + 2^{-tD(1-s\|\delta)}$ which, in turn, is no greater than $2 \cdot 2^{-tD(s\|\delta)}$ since $0 \leq s \leq \delta \leq 1/2$. The desired bound on τ now follows. The final statement in the theorem is just a consequence of the union bound. □

5 The noisy-network simulation protocol

Once the random inputs required in Π' have been generated, we essentially have a deterministic protocol, which can then be simulated on the noisy network as in [1]. We will now describe this simulation. For convenience, we will use Π to refer to the modified protocol Π'. Since we will have no

occasion to refer to the original protocol Π any more, this cannot result in any confusion.

For the purposes of the simulation, it will be necessary to artificially extend Π up to $2N$ steps. This can be done, e.g., by requiring each processor to transmit 0 on all its out-links in steps $N+1, \ldots, 2N$.

The simulation protocol will be called Σ. Σ proceeds in T *rounds*, indexed $t = 1, 2, \ldots, T$. Here, $T = 2N$. In each round, each processor decides either to simulate a step of Π, or to back up and cancel the last simulated step because of perceived errors in the progress of the simulation (how this decision is made will be explained later). In the former case, it transmits a bit (0 or 1) on each of its out-links, while in the latter case it transmits a special "back up" symbol $*$ on all its out-links.

Thus, in Σ, the processors communicate with each other using *three* symbols, viz., 0, 1, and $*$. Of course, these symbols must ultimately be encoded as bitstrings for transmission on the BSCs. Correspondingly, a decoding procedure for getting back the symbols from the bitstrings must be specified. One of the important ideas in [1] is to introduce *memory* into the encoding and decoding processes as a mechanism for recovering from errors caused by channel noise. The memory is introduced using a *tree code*, which we describe next.

5.1 The tree code

Consider a rooted complete ternary tree of depth T. The three edges out of each internal node will be understood to correspond to 0, 1, and $*$. A node at level t in this tree will be referred to by the sequence of t symbols (0, 1, or $*$) corresponding to the t edges leading to the node from the root. (The root itself will be referred to by the empty string Λ.)

Suppose each edge in the tree is labelled with a letter from a finite alphabet C, and this labelling satisfies the "relative Hamming distance $\geq 1/2$" condition, i.e., the Hamming distance between the sequences of labels along any two paths of length $l \geq 1$ that originate from a common node in the tree is at least $l/2$. Such a labelling constitutes a *ternary tree code* of depth T. The letters of C will be called *tree code characters.*

The surprising fact is that the size of the alphabet C required for such a labelling to exist does not depend on T. In fact, it can be shown (by a probabilistic existence argument) that an alphabet of size 1296 suffices to label the edges of a 3-ary tree of arbitrary depth T, so as to satisfy the above condition.

The processors agree upon such a ternary tree code of depth T before the simulation begins. Denote this code by $\mathcal{T}$. If $S(t-1) \in \{0,1,*\}^{t-1}$, $\sigma(t) \in \{0,1,*\}$, and $S(t) = S(t-1)\sigma(t)$ (concatenation), then let

$$\mathcal{T}\left(\sigma(t)\middle|S(t-1)\right) \in \mathcal{C}$$

be the tree code character labelling the edge between the nodes $S(t-1)$ and $S(t)$. Let $\mathcal{T}(S(t)) \in \mathcal{C}^t$ be the sequence of tree code characters labelling the t edges from the root to the node $S(t)$.

5.2 The block code

The processors also agree upon a *block code* of a suitably large blocklength k for the tree code characters, i.e., an encoding map $\chi_e : \mathcal{C} \to \{0,1\}^k$ and an associated decoding map $\chi_d : \{0,1\}^k \to \mathcal{C}$. It will turn out that k should exceed $K_1 \cdot \log(d+1)/C$, for some universal constant K_1. For purposes of analysis, we may assume that (χ_e, χ_d) is the *best* block code of blocklength k and size $|\mathcal{C}|$ for a BSC of crossover probability δ, i.e., of all such codes, it has the *least* worst-case probability of error (over all codewords).

5.3 Encoding procedure

We will now describe how processor q encodes the symbols 0, 1, and $*$ for transmission on the BSCs. Let $\sigma^{(q,r)}(t)$ be the symbol that q decides to transmit on the out-link (q,r) in round t, and let

$$S^{(q,r)}(t) = \left(\sigma^{(q,r)}(1), \ldots, \sigma^{(q,r)}(t)\right).$$

Let $S^{(q,r)}(0)$ be the empty string Λ. Then, q first encodes $\sigma^{(q,r)}(t)$ by the tree code character

$$C^{(q,r)}(t) = \mathcal{T}\left(\sigma^{(q,r)}(t)\middle|S^{(q,r)}(t-1)\right),$$

and then transmits $\chi_e\left(C^{(q,r)}(t)\right)$, the k-bit codeword for $C^{(q,r)}(t)$, on the link (q,r).

Thus, the bitstring into which the symbol to be transmitted on a given link is encoded depends on the symbols transmitted on that link in all the previous rounds.

5.4 Decoding procedure

In each round, q receives possibly corrupted versions of the bitstrings of length k that were transmitted on its in-links. q first decodes these bitstrings into tree code characters, by applying the decoding map χ_d of the block code. Let $\hat{C}^{(p,q)}(t)$ be the tree code character that q decodes on the in-link (p,q) in round t. Then, q tries to reconstruct $S^{(p,q)}(t)$ (the sequence of symbols p sent on the link (p,q) in the first t rounds) based solely on $\hat{C}^{(p,q)}(1),\ldots,\hat{C}^{(p,q)}(t)$. Let $\hat{S}^{(p,q)}(t) \in \{0,1,*\}^t$ denote this estimate.

The decoding rule is the following: q chooses $\hat{S}^{(p,q)}(t)$ to minimize the Hamming distance between $\mathcal{T}(\hat{S}^{(p,q)}(t))$ and $(\hat{C}^{(p,q)}(1),\ldots,\hat{C}^{(p,q)}(t))$. Ties are resolved arbitrarily.

Thus, the decoding decision on a given link depends on the receptions on that link in all the previous rounds also. However, note that $\hat{S}^{(p,q)}(t)$ need not be an extension of $\hat{S}^{(p,q)}(t-1)$; in fact, they could be very different from each other.

Two types of decoding errors can occur in round t on the link (p,q): a *block code error* is said to occur if $\hat{C}^{(p,q)}(t) \neq C^{(p,q)}(t)$, while a *tree code error* is said to occur if $\hat{S}^{(p,q)}(t) \neq S^{(p,q)}(t)$. Note that the occurence of one of these errors does not imply the occurence of the other: even if $\hat{C}^{(p,q)}(t) \neq C^{(p,q)}(t)$, $\hat{S}^{(p,q)}(t)$ can end up being equal to $S^{(p,q)}(t)$ if not too many of $\hat{C}^{(p,q)}(1),\ldots,\hat{C}^{(p,q)}(t-1)$ were wrong. Conversely, even if $\hat{C}^{(p,q)}(t) = C^{(p,q)}(t)$, $\hat{S}^{(p,q)}(t)$ may not be equal to $S^{(p,q)}(t)$ if too many of $\hat{C}^{(p,q)}(1),\ldots,\hat{C}^{(p,q)}(t-1)$ were wrong.

5.5 Transcripts and path estimates

Recall that in each round q either sends a bit (0 or 1) on each out-link, or sends $*$ on all its out-links (the first case corresponds to q simulating a step of Π, and the second to q cancelling the last simulated step); further, $S^{(q,r)}(t) \in \{0,1,*\}^t$ is the sequence of symbols sent by q in the first t rounds on the out-link (q,r).

Define $Tr(S^{(q,r)}(t))$ to be the bitstring obtained from $S^{(q,r)}(t)$ by the following rule: each $*$, going from left to right, erases the rightmost preceding un-erased 0 or 1. This rule reflects the fact that each $*$ represents a back up that cancels the last simulated step.

$Tr(S^{(q,r)}(t))$ will be called q's *out-transcript* for the link (q,r) after round t. If $B(q,t)$ is the number of times in the first t rounds that q backed up, then the length of $Tr(S^{(q,r)}(t))$ is obviously $t-2B(q,t)$ for each $r \in V_{out}(q)$.

This is the number of steps of Π that q thinks have been simulated at the end of round t. For this reason, $t - 2B(q,t)$ will be called the *apparent time* at q after t rounds, and denoted by $AT(q,t)$. Let $AT(q,0) = 0$. In round t, if q decides to simulate a step, that step would be $AT(q,t-1)+1$; if it decides to cancel a step, it would be step $AT(q,t-1)$.

Next, recall that $\hat{S}^{(p,q)}(t)$ is q's estimate of the sequence of symbols (0, 1, or $*$) that were transmitted on the in-link (p,q) in the first t rounds. Define $Tr(\hat{S}^{(p,q)}(t))$ to be the bitstring obtained from $\hat{S}^{(p,q)}(t)$ by the same rule as above. (If a tree code error occurs on the link (p,q) in round t, there may be a $*$ in $\hat{S}^{(p,q)}(t)$ for which there is no preceding un-erased 0 or 1. In that case, simply define $Tr(\hat{S}^{(p,q)}(t))$ to be the empty string Λ.) $Tr(\hat{S}^{(p,q)}(t))$ will be called q's *in-transcript* for the link (p,q) after t rounds. Note that the length of $Tr(\hat{S}^{(p,q)}(t))$ need not be the same for all $p \in V_{in}(q)$.

Now, the objective of processor q in the simulation is to learn the N bits it would have received on each of its in-links in Π, under the current assignment of input and random variable values. To this end, it maintains an estimate of some prefix of the N bits it would have received on each in-link. These will be called *path estimates*. Let $\hat{W}^{(p,q)}(t)$ denote q's path estimate for the in-link (p,q) at the end of round t.

In each round, q updates its path estimates either by extending each by one bit, or by erasing the last bit in each. Consequently, for any t, the length of $\hat{W}^{(p,q)}(t)$ is the same for all $p \in V_{in}(q)$. Denote this common length by $L(q,t)$. Define $\hat{W}^{(p,q)}(0)$ to be the empty string Λ for all $p \in V_{in}(q)$, so that $L(q,0) = 0$.

5.6 Outline of the simulation protocol

At this point, it will be useful to outline the simulation protocol from the perspective of processor q. In round t, q does the following in sequence and in synchronism with the other processors:

- First, it decides whether to simulate step $AT(q,t-1)+1$ or to back up and cancel step $AT(q,t-1)$, and computes $\sigma^{(q,r)}(t)$ for each $r \in V_{out}(q)$ accordingly. This will be explained in Section 5.8.

- It encodes each $\sigma^{(q,r)}(t)$ as a k-bit string, as described in Section 5.3, and transmits the bitstrings on the corresponding out-links.

- It decodes the k-bit string received on each in-link, as described in Section 5.4, to get $\hat{S}^{(p,q)}(t)$ for each $p \in V_{in}(q)$.

- Finally, it computes the updated path estimate $\hat{W}^{(p,q)}(t)$ for each $p \in V_{in}(q)$. This will be explained in Section 5.7.

5.7 Updating path estimates

In round t, q updates its path estimates based on a comparison between $\hat{W}^{(p,q)}(t-1)$ and $Tr(\hat{S}^{(p,q)}(t))$ for each $p \in V_{in}(q)$.

- If $\hat{W}^{(p,q)}(t-1)$ is a proper prefix of $Tr(\hat{S}^{(p,q)}(t))$ for all p, q concludes that it is in agreement with all its in-neighbors about the first $L(q,t-1)$ steps of Π. Therefore, it extends $\hat{W}^{(p,q)}(t-1)$ by bit $L(q,t-1)+1$ of $Tr(\hat{S}^{(p,q)}(t))$, to get $\hat{W}^{(p,q)}(t)$. In this case, $L(q,t) = L(q,t-1)+1$.

- If $\hat{W}^{(p,q)}(t-1)$ is not a proper prefix of $Tr(\hat{S}^{(p,q)}(t))$ for some p, q concludes that there is a disagreement with that in-neighbor about the first $L(q,t-1)$ steps of Π. Therefore, it erases the last bit (i.e., bit $L(q,t-1)$) of $\hat{W}^{(p,q)}(t-1)$ to get $\hat{W}^{(p,q)}(t)$. In this case, $L(q,t) = L(q,t-1)-1$.

There is a slight complication if $L(q,t-1) = 0$ (i.e., $\hat{W}^{(p,q)}(t-1) = \Lambda$ for all p) *and* $Tr(\hat{S}^{(p,q)}(t)) = \Lambda$ for some p. In this case, it will be convenient to define $\hat{W}^{(p,q)}(t)$ to be a special "bitstring" Λ', obtained by erasing the "last bit" of the empty string Λ. Correspondingly, $L(q,t)$ will be defined to be -1. (The purpose behind this convention is to ensure that $L(q,t) = L(q,t-1) \pm 1$ for all t.)

With this convention, it becomes necessary to specify what q should do in round t if $L(q,t-1) = -1$. In this case, q takes $\hat{W}^{(p,q)}(t)$ to be Λ for all i, regardless of what the in-transcripts are; correspondingly, $L(q,t)$ is 0.

5.8 Decision rule for simulating a step or backing up

To complete the description of the simulation, we must specify how q decides in round t whether to simulate a step or to back up. The decision is based on a comparison between $AT(q,t-1)$ and $L(q,t-1)$, i.e., the lengths of its out-transcripts and path estimates after the previous round.

- If $AT(q,t-1) = L(q,t-1)$, then q simulates step $AT(q,t-1)+1$. In this case, $\sigma^{(q,r)}(t)$ is the bit it would have transmitted on the link (q,r) in step $AT(q,t-1)+1$ of Π, had it received the sequence of bits $\hat{W}^{(p,q)}(t-1)$ on the in-link (p,q) in the previous steps.

- If $AT(q,t-1) \neq L(q,t-1)$, then q backs up and cancels step $AT(q,t-1)$. In this case, $\sigma^{(q,r)}(t)$ is $*$ for all $r \in V_{out}(q)$.

An easy induction on t proves that, for all $1 \leq t \leq T$, $L(q,t-1)$ equals either $AT(q,t-1)$ or $AT(q,t-1)-2$, and in either case $AT(q,t) = L(q,t-1)+1$. Also, for all $2 \leq t \leq T$, $AT(q,t) = AT(q,t-1)+1$ iff $L(q,t-1) = L(q,t-2)+1$. In other words, in round t, q simulates a step if it extended its path estimates in round $t-1$, and backs up if it shortened them in round $t-1$.

5.9 Final decision

After round T of the simulation, q simply checks if $L(q,T)$, the length of each $\hat{W}^{(p,q)}(T)$, is at least N. If so, it concludes that the simulation was successful, and takes the first N bits of $\hat{W}^{(p,q)}(T)$ to be the bits it would have received on the in-link (p,q) in Π. Otherwise, it concludes that the simulation failed. This completes the description of the simulation protocol Σ.

It remains to lower bound the probability that all the processors succeed in computing the right bits at the end of the simulation.

6 Analysis

6.1 Remarks

We will first define the key measure of the progress made by the simulation at processor q after t rounds: $RP(q,t)$, the *real progress* at q after t rounds, is the largest $n \leq L(q,t)$ such that, for all $p \in V_{in}(q)$, the first n bits of $\hat{W}^{(p,q)}(t)$ are "correct," i.e., they equal the bits that q would have received on the in-link (p,q) in the first n steps of Π (under the current assignment of input and random variable values). If $L(q,t) = 0$ or -1, then $R(q,t)$ is set equal to $L(q,t)$. Note that $RP(q,t)$, unlike $AT(q,t)$ or $L(q,t)$, cannot be calculated by q during the simulation. From the definition, it is obvious that:

Observation 6.1 *The simulation is successful iff $RP(q,T) \geq N$ for all $q \in V$.*

It should be pointed out here that $AT(q,t)$, $L(q,t)$, and $RP(q,t)$ can exceed N, since Π was artificially extended to $2N$ steps.

Define $\mathcal{H}(q,t)$, the *history-cone* at q after t rounds, to be the set of all processor-time pairs (p,τ) such that there exists a directed path in G, from p to q, of length at most $t-\tau$. Note that $(q,t) \in \mathcal{H}(q,t)$, and if $(p,t) \in \mathcal{H}(q,t)$ then $p = q$. Intuitively, if $(p,\tau) \in \mathcal{H}(q,t)$ and $\tau < t$, then a signal from p in round τ will reach q in round $t-1$ or earlier, and will therefore affect q's actions in round t. (q,t) is included in $\mathcal{H}(q,t)$ for technical convenience.

A *time-like* sequence in $\mathcal{H}(q,t)$, of length m, is a sequence

$$(p_1,t_1),(p_2,t_2),\ldots,(p_m,t_m),$$

where $1 \leq t_1 < t_2 < \cdots < t_m \leq t$, $(p_k,t_k) \in \mathcal{H}(p_{k+1},t_{k+1})$ for $1 \leq k < m$, and $(p_m,t_m) \in \mathcal{H}(q,t)$.

Define $X(q,t)$ to be the largest m for which there exists a time-like sequence $(p_1,t_1),(p_2,t_2),\ldots,(p_m,t_m)$ in $\mathcal{H}(q,t)$ such that, for each $1 \leq k \leq m$, a tree code error occured on some in-link at processor p_k in round t_k. (If there is no such sequence, let $X(q,t) = 0$.) Note that all these tree-code errors propagate to q by round $t-1$ or earlier, and will therefore affect its actions in round t. Let $X(q,0) = 0$.

The following simple results will be used freely without reference in the sequel:

1. If $p = q$ or $p \in V_{in}(q)$, then $\mathcal{H}(p,t-1) \subset \mathcal{H}(q,t)$ and, consequently, $X(q,t) \geq X(p,t-1)$. Further, if q makes a tree code error in round t, then $X(q,t) \geq X(p,t-1)+1$.

2. $RP(q,t-1)$ and $RP(q,t)$ can differ by at most 1. $RP(q,t) = RP(q,t-1)+1$ only if $RP(q,t-1) = L(q,t-1)$ and $L(q,t) = L(q,t-1)+1$. Similarly, $RP(q,t) = RP(q,t-1)-1$ only if $RP(q,t-1) = L(q,t-1)$ and $L(q,t) = L(q,t-1)-1$.

3. For any t, the first $RP(q,t-1)+1$ bits of each of q's out-transcripts after t rounds are "correct" (i.e., they are the same as in Π).

6.2 Outline of analysis

The following theorem is the main result of [1]:

Theorem 6.1 *There exist universal constants $K_1, K_2 > 0$ (independent of the protocol Π and the network G) such that the failure probability of the simulation under any assignment of inputs and random variable values is bounded above by*

$$|V| \cdot \exp\{-K_2\,(kC - K_1 \log(d+1))\,N\},$$

provided k, the blocklength of the block code used, exceeds $(K_1/C)\cdot\log(d+1)$. Here, d is the maximum degree of any node in G, and $C = 1 - h(\delta)$ is the capacity of each link in the network.

There are two main lemmas, Lemma 6.1 and Lemma 6.2, in the proof of Theorem 6.1. The first states that if $RP(q,t)$ is small relative to t (i.e., q has not progressed very much after round t), then $X(q,t)$ must be correspondingly large (i.e., there must have been many tree-code errors in the first t rounds that affected the progress of q).

Lemma 6.1 *If $RP(q,t) = t - l$, then $X(q,t) \geq l/2$. Equivalently,*

$$RP(q,t) \geq t - 2X(q,t).$$

The second lemma upper bounds the probability that $X(q,t) > t/4$ for some q. It states that, if k (the blocklength of the block code used) is large enough, the above probability will decay exponentially in t.

Lemma 6.2 *There exist universal constants $K_1, K_2 > 0$ such that, if $k > (K_1/C)\cdot\log(d+1)$, the probability that $X(q,t) > t/4$ for some q is bounded above by*

$$|V| \cdot \exp\left\{-K_2\left(kC - K_1\log(d+1)\right)t/2\right\}.$$

By Observation 6.1, Lemma 6.1, and the fact that $T = 2N$, the probability that the simulation fails is upper bounded by the probability that $X(q,T) > T/4$ for some q. Applying Lemma 6.2 now, with $t = T$, gives the result of Theorem 6.1.

The proof of Lemma 6.2 is exactly as in [1] and will not be repeated here. Lemma 6.1, however, is slightly different from its counterpart in [1], and will therefore be proved fully. The proof is based on the same ideas as in [1].

6.3 Proof of the first lemma

We will actually prove a stronger result, viz., for all $0 \leq t \leq T$ and all $q \in V$,

$$2RP(q,t) + 2X(q,t) - L(q,t) \geq t. \tag{6}$$

If (6) holds, then

$$\begin{aligned} RP(q,t) - t + 2X(q,t) &\geq L(q,t) - RP(q,t) \\ &\geq 0, \end{aligned}$$

thus proving Lemma 6.1. The proof of (6) is by induction on t. It obviously holds for $t = 0$. So, assume that $t \geq 1$, and that

$$2RP(q, t-1) + 2X(q, t-1) - L(q, t-1) \geq t - 1, \tag{7}$$

for all $q \in V$. The induction step requires considering six disjoint and exhaustive events at processor q in round t. They are:

1. q makes a tree code error in round t.
   (a) $L(q,t) = L(q,t-1) - 1$.
   (b) $L(q,t) = L(q,t-1) + 1$.
2. q does not make a tree code error in round t.
   (a) $L(q,t) = L(q,t-1) - 1$.
      i. $RP(q,t) < RP(q,t-1)$.
      ii. $RP(q,t) \geq RP(q,t-1)$.
   (b) $L(q,t) = L(q,t-1) + 1$.
      i. $RP(q,t) \leq RP(q,t-1)$.
      ii. $RP(q,t) > RP(q,t-1)$.

6.3.1 Cases 1.a, 1.b, 2.a.ii, 2.b.ii

These four cases are easy to handle. In Case 1.a, $RP(q,t) \geq RP(q,t-1)-1$, $X(q,t) \geq X(q,t-1)+1$, and $L(q,t) = L(q,t-1)-1$. In Case 1.b, $RP(q,t) \geq RP(q,t-1)$, $X(q,t) \geq X(q,t-1)+1$, and $L(q,t) = L(q,t-1)+1$. In Case 2.a.ii, $RP(q,t) = RP(q,t-1)$, $X(q,t) \geq X(q,t-1)$, and $L(q,t) = L(q,t-1)-1$. Finally, in Case 2.b.ii, $RP(q,t) = RP(q,t-1)+1$, $X(q,t) \geq X(q,t-1)$, and $L(q,t) = L(q,t-1)+1$. Thus, in all these cases,

$$\begin{aligned} & 2RP(q,t) + 2X(q,t) - L(q,t) \\ & \quad \geq \; 2RP(q,t-1) + 2X(q,t-1) - L(q,t-1) + 1 \\ & \quad \geq \; t, \end{aligned}$$

by the induction hypothesis at q. This proves (6).

The other two cases, viz. 2.a.i and 2.b.i, are more subtle; they require an examination of events at the in-neighbors of q also.

6.3.2 Case 2.a.i

Here, $RP(q,t) = RP(q,t-1) - 1$, $X(q,t) \geq X(q,t-1)$, and $L(q,t) = L(q,t-1) - 1$. This is not enough to complete the induction step. However, we claim that:

Claim 6.1 *In Case 2.a.i, there exists a $p \in V_{in}(q)$ for which $RP(p,t-1) < RP(q,t)$.*

Proof: If $L(p,t-1) < L(q,t-1)$ for some $p \in V_{in}(q)$ then, in fact, $L(p,t-1) < L(q,t-1) - 1$, since $L(p,t-1)$ and $L(q,t-1)$ are either both odd or both even. But $RP(p,t-1) \leq L(p,t-1)$ and $L(q,t-1) - 1 = RP(q,t)$, so that we have $RP(p,t-1) < RP(q,t)$.

So, assume that $L(p,t-1) \geq L(q,t-1)$ for all $p \in V_{in}(q)$. If $RP(p,t-1) \geq RP(q,t)$ for all $p \in V_{in}(q)$, then the first $RP(q,t)+1 = RP(q,t-1)$ bits of $Tr(S^{(p,q)}(t))$ would be "correct" for all $p \in V_{in}(q)$. But $Tr(S^{(p,q)}(t)) = Tr(\hat{S}^{(p,q)}(t))$, since q did not make any tree code errors in round t. Therefore, for all $p \in V_{in}(q)$, (a) the length of $Tr(\hat{S}^{(p,q)}(t))$ would be $AT(p,t) = L(p,t-1)+1 \geq L(q,t-1)+1$ by assumption, and (b) the first $RP(q,t-1) = L(q,t-1)$ bits of $Tr(\hat{S}^{(p,q)}(t))$ would be "correct" and hence equal to $\hat{W}^{(p,q)}(t-1)$. But, under these conditions, q would have extended its path estimates in round t, and so we have a contradiction. This proves that there must exist a $p \in V_{in}(q)$ for which $RP(p,t-1) < RP(q,t)$. □

Now, the $p \in V_{in}(q)$ provided by the above claim satisfies $RP(q,t) \geq RP(p,t-1)+1$, $2X(q,t) \geq 2X(p,t-1)$, and $0 \geq RP(p,t-1) - L(p,t-1)$. Further, we have $RP(q,t) - L(q,t) = 0$. Adding these four inequalities, we get

$$\begin{aligned} & 2RP(q,t) + 2X(q,t) - L(q,t) \\ & \geq \; 2RP(p,t-1) + 2X(p,t-1) - L(p,t-1) + 1 \\ & \geq \; t, \end{aligned}$$

by the induction hypothesis at p. This proves (6).

6.3.3 Case 2.b.i

Here, $RP(q,t) = RP(q,t-1)$, $X(q,t) \geq X(q,t-1)$, and $L(q,t) = L(q,t-1) + 1$. Again, this is not enough to complete the induction step. However, we claim that:

Claim 6.2 *In Case 2.b.i,*

1. $L(p, t-1) \geq L(q, t-1)$ *for all* $p \in V_{in}(q)$.

2. *There exists a* $p \in V_{in}(q)$ *for which* $RP(p, t-1) < RP(q, t)$.

Proof: Consider any $p \in V_{in}(q)$. Since q extended its path estimates in round t, $\hat{W}^{(p,q)}(t-1)$ must be a proper prefix of $Tr(\hat{S}^{(p,q)}(t))$. In particular, $L(q, t-1)$ is smaller than the length of $Tr(\hat{S}^{(p,q)}(t))$. But $Tr(\hat{S}^{(p,q)}(t)) = Tr(S^{(p,q)}(t))$, since q did not make any tree code errors in round t, and so the length of $Tr(\hat{S}^{(p,q)}(t))$ is $AT(p, t) = L(p, t-1) + 1$. Thus, $L(q, t-1) < L(p, t-1) + 1$, proving the first assertion.

If $RP(p, t-1) \geq RP(q, t)$ for all $p \in V_{in}(q)$, then the first $RP(q, t)+1$ bits of $Tr(S^{(p,q)}(t))$ would be "correct" for all $p \in V_{in}(q)$. Since $Tr(S^{(p,q)}(t)) = Tr(\hat{S}^{(p,q)}(t))$, and $\hat{W}^{(p,q)}(t)$ is a prefix of $Tr(\hat{S}^{(p,q)}(t))$ of length $L(q, t) \geq RP(q, t)+1$, this would imply that the first $RP(q, t)+1$ bits of each $\hat{W}^{(p,q)}(t)$ are "correct." But this contradicts the definition of $RP(q, t)$. Thus, there must exist a $p \in V_{in}(q)$ for which $RP(p, t-1) < RP(q, t)$. This proves the second assertion. □

Now, for the $p \in V_{in}(q)$ provided by the second assertion of the last claim, we have $RP(q, t) \geq RP(p, t-1) + 1$, $X(q, t) \geq X(p, t-1)$, and $L(q, t) \leq L(p, t-1) + 1$. Therefore,

$$\begin{aligned} & 2RP(q,t) + 2X(q,t) - L(q,t) \\ & \quad \geq \; 2RP(p,t-1) + 2X(p,t-1) - L(p,t-1) + 1 \\ & \quad \geq \; t, \end{aligned}$$

by the induction hypothesis at p. This proves (6).

References

[1] S. Rajagopalan and L.J. Schulman. A coding theorem for distributed computation. *Proc. of the 26th Annual ACM Symposium on the Theory of Computing*, 1994.

[2] L.J. Schulman. Deterministic coding for interactive communication. *Proc. of the 25th Annual ACM Symposium on the Theory of Computing*, 1993.

[3] S. Venkatesan and V. Anantharam. Deterministic simulation of randomized protocols over noisy channels. Technical Report UCB/ERL M94/102, Electronics Research Laboratory, Univ. of California, Berkeley, December 1994.