DETERMINISTIC SIMULATION OF RANDOMIZED PROTOCOLS OVER NOISY CHANNELS

by

S. Venkatesan and V. Anantharam

Memorandum No. UCB/ERL M94/102

30 December 1994

COVER PAGE

DETERMINISTIC SIMULATION OF RANDOMIZED PROTOCOLS OVER NOISY CHANNELS

by

S. Venkatesan and V. Anantharam

Memorandum No. UCB/ERL M94/102

30 December 1994

ELECTRONICS RESEARCH LABORATORY

College of Engineering
University of California, Berkeley
94720

DETERMINISTIC SIMULATION OF RANDOMIZED PROTOCOLS OVER NOISY CHANNELS

by

S. Venkatesan and V. Anantharam

Memorandum No. UCB/ERL M94/102

30 December 1994

ELECTRONICS RESEARCH LABORATORY

College of Engineering
University of California, Berkeley
94720

Deterministic simulation of randomized protocols over noisy channels[*]

S. Venkatesan[†] V. Anantharam[‡§]

Abstract

Suppose the input to a function f is split between two processors connected by noiseless binary channels. The *communication complexity* of f measures the number of bits they must exchange to compute f. Since its introduction by Yao, this interactive model of computation has been widely studied, with the objective of characterizing the inherent cost of communication in distributed computation. Since all practical channels are noisy, it is of interest to study the effect of channel noise on the complexity and reliability of this communication. In this direction, L.J. Schulman recently proved that: any noiseless-channel deterministic protocol of complexity L that computes f correctly can be deterministically simulated over noisy channels with $O(L)$ transmissions, while incurring error probability $2^{-\Omega(L)}$ in computing f. However, Schulman's result can be strengthened. We prove that: any noiseless-channel *randomized* protocol of complexity L which computes f with error probability ϵ can be *deterministically* simulated over noisy channels using $O(L)$ transmissions, while incurring error probability $\epsilon + 2^{-\Omega(L)}$ in computing f. During the simulation, the two processors generate the randomness they need *using channel noise*, so that, in a certain sense, they turn noise to their advantage. This result is significant because allowing randomization and a small probability of error can substantially reduce the (noiseless-channel) communication complexity of certain functions.

Index Terms:

Communication complexity, randomized protocols, noisy-channel simulation of protocols, generating randomness from noise.

[*]Research supported by NSF IRI 9005849, IRI 9310670, and NCR 8857731.

[†]Cornell University and U.C. Berkeley.

[‡]Univ. of California, Berkeley. The major portion of this work was completed when the author was affiliated with Cornell University.

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

1 Introduction

In [4], A.C. Yao introduced the notion of *communication complexity* in connection with the following problem:

> Suppose S_X, S_Y, and W are finite sets, and $f : S_X \times S_Y \rightarrow W$. Two processors know $x \in S_X$ and $y \in S_Y$ respectively, and both wish to know $f(x, y)$. To this end, they communicate with each other over a pair of binary channels. How many transmissions do they need to compute f, given any pair of arguments?

Roughly speaking, the number of transmissions needed is the communication complexity of the function.

Yao's original motivation for studying this problem was to provide a framework to understand the significance of communication in distributed computation. Since then, it has been shown to have implications for other problems, like time-area tradeoffs in VLSI and depths of Boolean circuits. (See [1] or [2] for a survey of research in this area.)

In all these investigations, the channels connecting the two processors have been assumed to be noiseless. However, all practical channels are noisy, and it is therefore important to study the effect of channel noise on the complexity and reliability of the communication between the two processors. One particularly interesting question is the following: is it possible to simulate noiseless-channel protocols over noisy channels (through appropriate coding techniques) with only a constant factor overhead in the number of transmissions, if a small error probability in the computation can be tolerated? The analogous question in the case of the data transmission problem (one-way communication) was, of course, addressed by Shannon in 1948, resulting in his famous coding theorem. Note that Shannon's theorem cannot be directly applied here, because of the *interactive* nature of the exchange of information: in general, neither processor knows in advance all its future transmissions, and therefore cannot code large blocks of data as in the data transmission case. Recently, however, L.J. Schulman [3] proved the following result in this direction:

Theorem 1.1 *Assume that f can be correctly computed by two processors, using at most L transmissions on any pair of arguments, when the channels connecting them are noiseless. Suppose the two channels are actually binary symmetric channels (BSCs), each with crossover probability p ($0 < p < 1/2$). Then, it is possible for the two processors to compute f on any pair of arguments with an error probability of at most 2^{-L}, while using no more than $KL/(1-h(p))$ transmissions in all. Here, K is a universal constant, and $h(p) = -p\log_2 p - (1-p)\log_2(1-p)$.*

Thus, long enough protocols can indeed be simulated with a constant factor overhead while incurring an arbitrarily small error probability (a result analogous to Shannon's theorem). The key idea in the proof is the use of *tree codes* by both processors, as a mechanism for recovering from errors caused by channel noise.

Now, in the noiseless-channel case, it is known that allowing the processors access to independent sources of *randomness* and tolerating a small probability of error in the computation can significantly reduce the communication complexity of certain functions. For example [2], consider the decision problem of determining whether or not two bitstrings of length n are equal (when each processor knows one of them, to begin with). Any deterministic error-free solution requires at least $n+1$ transmissions on some pair of inputs. However, if even one of the processors has a source of random bits, it is possible to solve the problem with $O(\log n)$ transmissions on any pair of inputs (provided, of course, that a small error probability is allowed).

Therefore, it is natural to ask whether a result in the same spirit as Schulman's can be proved for the simulation of private-coin *randomized* protocols over noisy channels. This is the subject of the present paper. Somewhat surprisingly, it turns out that *long enough randomized protocols can be* **deterministically** *simulated over noisy channels with only a constant factor overhead in the number of transmissions, while incurring an error probability that is arbitrarily close to that of the given protocol.* No restrictions are placed on the sources of randomness that the two processors use in the original randomized protocol. This strengthens Schulman's result (which, however, is used in an essential way here). An interesting feature of the result is that the processors use channel noise as a source of randomness, thus turning it to their advantage in a certain sense. A more precise statement of our result is given below:

Theorem 1.2 *Let ϕ be a given noiseless-channel private-coin randomized protocol of complexity L, which computes the function f with an error probability $< \epsilon$ on any pair of arguments. Now, suppose the two channels are actually BSCs, each with crossover probability p $(0 < p < 1/2)$. Then, for every $\alpha > 1$, $\beta > 1$, and $\gamma > 8448$, there exists a way of* **deterministically** *simulating ϕ over the BSCs, using $2\lceil \beta \lceil \alpha L \rceil / h(p) \rceil + 4 \lceil \gamma/(1-h(p)) \rceil L$ transmissions in all, and incurring an error probability on any pair of arguments of at most $\epsilon + 2P_e(\alpha,\beta,\gamma,L,p)$, where $P_e(\alpha,\beta,\gamma,L,p)$ equals*

$$\min_{\substack{0<s<p \\ h(s)>h(p)/\beta}} \left\{ 2^{-\beta\alpha L \frac{D(s\|p)}{h(p)}} + 2^{-\beta\alpha L \frac{D(1-s\|p)}{h(p)}} + (\beta \lceil \alpha L \rceil / h(p))\, 2^{-\frac{\alpha L}{h(p)}[\beta h(s) - h(p)]} \right\} + 2^{-(\alpha-1)L} + 2^{-\left(\frac{\gamma-8448}{2048}\right)L}.$$

Here,

$$D(u\|v) = u \log_2 \frac{u}{v} + (1-u) \log_2 \left(\frac{1-u}{1-v}\right).$$

Here is a rough outline of the proof:

In Section 3, we show how an arbitrary private-coin randomized protocol ϕ can be modified to another one, ψ, of the same complexity L, in which each processor uses only $R = \lceil \alpha L \rceil$ independent and unbiased bits for randomization (for any $\alpha > 1$).

The price to be paid for this is a slight increase in error probability (exponentially small in αL).

Then, each of the processors generates the required number of random bits *using channel noise*, by a deterministic process. In Section 4, we show that R independent and unbiased bits can be deterministically generated, using $\lceil \beta R/h(p) \rceil$ transmissions over a BSC of crossover probability $0 < p < 1/2$ (for any $\beta > 1$), with a failure probability that decreases exponentially in βR.

Once the required random inputs have been generated, we essentially have a deterministic protocol of complexity L, to which Schulman's result applies. In its original formulation, this result is proved for protocols in which all messages are one bit long. (This is not a significant restriction, since any protocol can be artificially converted to a "bit-by-bit" one with at most double the complexity.) However, in Section 5, we modify the proof to directly apply to arbitrary protocols and, in the process, also correct certain (minor) errors in the original proof.

But first, in Section 2, we introduce all the required definitions and notation.

2 Preliminaries

As in Section 1, let S_X, S_Y, and W be finite sets, and $f : S_X \times S_Y \to W$. Processors A_X and A_Y wish to compute f by communicating across a pair of binary channels, when each of them knows one of the arguments of f. The communication proceeds according to some agreed-upon *protocol*, i.e., a set of rules that determines, at each stage, which processor is to transmit, and what that processor's transmission should be, based on all the information available to it until then. Protocols can be either deterministic or randomized. In this section, we will formally define *private-coin randomized protocols* (of which deterministic protocols are degenerate special cases).

In a private-coin randomized protocol ϕ, A_X and A_Y have access to separate and independent sources of randomness, i.e., A_X knows $\omega_X \in \Omega_{\phi,X}$ drawn according to the p.d. $P_{\phi,X}$, and A_Y knows $\omega_Y \in \Omega_{\phi,Y}$ drawn independently according to the p.d. $P_{\phi,Y}$. We will use P_ϕ to represent the product p.d. $P_{\phi,X} \cdot P_{\phi,Y}$ on $\Omega_{\phi,X} \times \Omega_{\phi,Y}$. They can then base their transmissions not only on their respective arguments (x or y) and on all past transmissions, but also on their respective random inputs. Since S_X and S_Y are finite sets and the protocol is required to terminate in finitely many steps, there is no loss of generality in assuming that $\Omega_{\phi,X}$ and $\Omega_{\phi,Y}$ are also finite, but arbitrarily large, sets. $P_{\phi,X}$ and $P_{\phi,Y}$ are arbitrary probability distributions.

We need the following preliminary definitions: let $\mathcal{X}_\phi = S_X \times \Omega_{\phi,X}$ and $\mathcal{Y}_\phi = S_Y \times \Omega_{\phi,Y}$. We will refer to elements of S_X and S_Y as *arguments*, to elements of $\Omega_{\phi,X}$ and $\Omega_{\phi,Y}$ as *random inputs*, and to elements of $\mathcal{X}_\phi$ and $\mathcal{Y}_\phi$ simply as *inputs*. By a *rectangle*, we mean a product set of the form $\mathcal{F} = F_X \times F_Y$, where $F_X \subseteq \mathcal{X}_\phi$ and $F_Y \subseteq \mathcal{Y}_\phi$. F_X and F_Y are respectively the *X- and Y- projections* of $\mathcal{F}$. The rectangles $\mathcal{G} = G_X \times G_Y$ and $\mathcal{H} = H_X \times H_Y$ are said to *X-partition* $\mathcal{F}$, if F_X is the disjoint union of G_X and H_X, and $F_Y = G_Y = H_Y$. (Note that $\mathcal{F}$ itself is the disjoint union of $\mathcal{G}$ and $\mathcal{H}$.) Y-partitions are analogously defined.

The protocol ϕ can be described in terms of an associated rooted binary tree T_ϕ. The nodes of T_ϕ are all rectangles, with the root being $\mathcal{X}_\phi \times \mathcal{Y}_\phi$. Each internal node is either X- or Y-partitioned by its children (and is thus the disjoint union of its children). All left (resp. right) edges are labelled by 0 (resp. 1). Each leaf $\mathcal{F}$ is marked with an element of W which is called the *computed value* associated with that leaf (denoted by $V_\phi(\mathcal{F})$). The root is at level 0, its children are at level 1, etc.. Clearly, all the rectangles at any given level must be disjoint. Further, the leaves of the tree must all be disjoint, and their union must be $\mathcal{X}_\phi \times \mathcal{Y}_\phi$.

For each pair of inputs (x, ω_X) and (y, ω_Y), the protocol associates a unique path in the tree from the root to some leaf, and a corresponding sequence of transmitters and transmissions, in the following way: initially, A_X and A_Y are at the root, and after k transmissions, they are at some node at level k. If this node is Z-partitioned by its children (here, Z is either X or Y), then the next transmission is by A_Z. This transmission is 0 if the input of A_Z is contained in the Z-projection of the *left* child of the node (in this case, the processors move next to the left child of the current node), and is 1 otherwise (in this case, the processors move next to the right child of the current node). The protocol ends when a leaf is reached. Both processors then take the computed value associated with that leaf (denoted by $V_\phi(x, \omega_X, y, \omega_Y)$) as the value of the function f.

Clearly, the root-to-leaf path taken by the two processors is the *unique* one every node along which contains the pair of inputs $((x, \omega_X), (y, \omega_Y))$. The total number of transmissions, $L_\phi(x, \omega_X, y, \omega_Y)$, equals the depth of the leaf reached. The probability, $\epsilon_\phi(f; x, y)$, that ϕ errs in computing $f(x, y)$ is $P_\phi\{V_\phi(x, \omega_X, y, \omega_Y) \neq f(x, y)\}$.

We define the *complexity* of the protocol ϕ as

$$L_\phi = \max_{(x, \omega_X, y, \omega_Y)} L_\phi(x, \omega_X, y, \omega_Y)$$

and its *probability of error* in computing f as

$$\epsilon_\phi(f) = \max_{(x,y)} \epsilon_\phi(f; x, y).$$

The *ϵ-randomized complexity* of f is then defined as

$$C_R(f, \epsilon) = \min\{L_\phi : \epsilon_\phi(f) \leq \epsilon\}.$$

Note that we must have $\epsilon < 1 - |W|^{-1}$ in order to have functions with nontrivial complexities. For, if $\epsilon \geq 1 - |W|^{-1}$, then A_X can simply pick an element of W according to a uniform distribution, and transmit it to A_Y. Both can then take this element to be the value of the function being computed, thus incurring an error probability of only $1 - |W|^{-1}$.

3 From ϕ to ψ

Let ϕ be a private-coin randomized protocol of complexity L, which computes f with error probability ϵ (as defined in Section 2). In general, the sources of randomness associated with ϕ, $(\Omega_{\phi,X}, P_{\phi,X})$ and $(\Omega_{\phi,Y}, P_{\phi,Y})$, are arbitrary. In this section, we show that ϕ can be modified to a new private-coin randomized protocol, ψ, of the same complexity L, such that $\Omega_{\psi,X} = \Omega_{\psi,Y} = \{0,1,\ldots,2^R-1\}$, and $P_{\psi,X}$ and $P_{\psi,Y}$ are uniform distributions on $\{0,1,\ldots,2^R-1\}$. (Here, R is any integer $\geq L$.) Thus, in ψ, each processor uses R independent and unbiased bits for randomization. We will refer to such protocols as "R-uniform." The price one pays for this "uniformization" is a possible increase in the error probability.

Theorem 3.1 *Let $R = \lceil \alpha L \rceil$ (for some $\alpha > 1$). There exists an R-uniform randomized protocol ψ of the same complexity L as ϕ, such that*

$$\epsilon_\psi(f) \leq \epsilon + 2 \cdot 2^{-(\alpha-1)L}.$$

Proof: The idea of the proof is to approximate all relevant probabilities in the protocol ϕ by dyadic rationals with denominator 2^R. This is made precise below.

For each $x \in S_X$, $y \in S_Y$, and each node $\mathcal{F} = F_X \times F_Y$ in T_ϕ (the binary tree representing ϕ), let

$$\begin{aligned}\lambda_X(x,\mathcal{F}) &= P_{\phi,X}\{\omega_X : (x,\omega_X) \in F_X\},\\ \lambda_Y(y,\mathcal{F}) &= P_{\phi,Y}\{\omega_Y : (y,\omega_Y) \in F_Y\}.\end{aligned}$$

Note that, during the execution of ϕ on the arguments x and y, the probability that A_X and A_Y pass through the node $\mathcal{F}$ is just $\lambda_X(x,\mathcal{F}) \cdot \lambda_Y(y,\mathcal{F})$.

Next, for each $x \in S_X$ and each node $\mathcal{F}$ in T_ϕ, A_X defines an interval $[a_X(x,\mathcal{F}), b_X(x,\mathcal{F}))$ contained in $[0,1)$, in the following way:

a) If $\mathcal{F}$ is the root, then

$$[a_X(x,\mathcal{F}), b_X(x,\mathcal{F})) = [0,1).$$

b) If $\mathcal{F}$ is any node other than the root which is the *left* child of its parent ($\mathcal{G}$, say), then

$$\begin{aligned}a_X(x,\mathcal{F}) &= a_X(x,\mathcal{G}),\\ b_X(x,\mathcal{F}) &= a_X(x,\mathcal{G}) + \lambda_X(x,\mathcal{F}).\end{aligned}$$

c) If $\mathcal{F}$ is any node other than the root which is the *right* child of its parent $\mathcal{G}$, then

$$\begin{aligned}b_X(x,\mathcal{F}) &= b_X(x,\mathcal{G}),\\ a_X(x,\mathcal{F}) &= b_X(x,\mathcal{G}) - \lambda_X(x,\mathcal{F}).\end{aligned}$$

Finally, A_X defines $M_X(x,\mathcal{F})$ and $N_X(x,\mathcal{F})$ to be the unique integers m and n satisfying

$$\frac{m-1/2}{2^R} \leq a_X(x,\mathcal{F}) < \frac{m+1/2}{2^R},$$

$$\frac{n-1/2}{2^R} \leq b_X(x,\mathcal{F}) < \frac{n+1/2}{2^R}.$$

If $\mathcal{F}$ is the root, then $M_X(x,\mathcal{F}) = 0$ and $N_X(x,\mathcal{F}) = 2^R$. Also, if $\lambda_X(x,\mathcal{F})$ is sufficiently small, $M_X(x,\mathcal{F})$ could equal $N_X(x,\mathcal{F})$.

In an analogous manner, A_Y defines intervals $[a_Y(y,\mathcal{F}), b_Y(y,\mathcal{F}))$ and integers $M_Y(y,\mathcal{F})$ and $N_Y(y,\mathcal{F})$, for each $y \in S_Y$ and each node $\mathcal{F}$ in T_ϕ.

Note that

$$\begin{aligned}\lambda_X(x,\mathcal{F}) &= b_X(x,\mathcal{F}) - a_X(x,\mathcal{F}),\\ \lambda_Y(y,\mathcal{F}) &= b_Y(y,\mathcal{F}) - a_Y(y,\mathcal{F}),\end{aligned}$$

for all x, y, and $\mathcal{F}$.

Also, if $\mathcal{F}$ is an internal node which is X-partitioned by its left child $\mathcal{G}$ and right child $\mathcal{H}$, then

$$\begin{aligned}M_X(x,\mathcal{F}) &= M_X(x,\mathcal{G}),\\ N_X(x,\mathcal{G}) &= M_X(x,\mathcal{H}),\\ N_X(x,\mathcal{H}) &= N_X(x,\mathcal{F}),\end{aligned}$$

so that the (possibly empty) set of integers

$$\{k : M_X(x,\mathcal{F}) \le k < N_X(x,\mathcal{F})\}$$

is the disjoint union of

$$\{k : M_X(x,\mathcal{G}) \le k < N_X(x,\mathcal{G})\}$$

and

$$\{k : M_X(x,\mathcal{H}) \le k < N_X(x,\mathcal{H})\}.$$

An analogous statement holds if $\mathcal{F}$ is Y-partitioned by its children.

We are now in a position to describe the R-uniform protocol ψ. The binary tree T_ψ associated with ψ is *isomorphic* to T_ϕ, i.e., it has the same form as T_ϕ. Further, if an internal node in T_ϕ is X-partitioned (resp. Y-partitioned) by its children, then the corresponding node in T_ψ is also X-partitioned (resp. Y-partitioned) by its children, so that the same processor transmits at corresponding nodes. Recall that the nodes of T_ψ must be rectangles contained in $\mathcal{X}_\psi \times \mathcal{Y}_\psi$, where

$$\begin{aligned}\mathcal{X}_\psi &= S_X \times \left\{0,1,\ldots,2^R-1\right\},\\ \mathcal{Y}_\psi &= S_Y \times \left\{0,1,\ldots,2^R-1\right\}.\end{aligned}$$

The rule for obtaining the nodes of T_ψ is this: replace each node $\mathcal{F}$ in T_ϕ by the rectangle $\mathcal{U} = U_X \times U_Y$, where U_X equals

$$\bigcup_x \{x\} \times \{k : M_X(x,\mathcal{F}) \le k < N_X(x,\mathcal{F})\}$$

and U_Y equals

$$\bigcup_y \{y\} \times \{k : M_Y(y,\mathcal{F}) \le k < N_Y(y,\mathcal{F})\}.$$

Note that $\mathcal{U}$ could be empty if $\lambda_X(x,\mathcal{F}) \cdot \lambda_Y(y,\mathcal{F})$ is sufficiently small for all (x,y).

Finally, each leaf in T_ψ has the same computed value as the corresponding leaf in T_ϕ.

It is easily verified that this yields an R-uniform protocol. Clearly, ψ has the same complexity as ϕ. It only remains to estimate the error probability that ψ incurs in computing f.

Fix any pair of arguments (x,y). Let $\mathcal{U}$ be any leaf of T_ψ, and $\mathcal{F}$ the corresponding leaf of T_ϕ. Let $p_\phi(\mathcal{F},x,y)$ (resp. $p_\psi(\mathcal{U},x,y)$) be the probability that, in the protocol ϕ (resp. ψ), A_X and A_Y reach the leaf $\mathcal{F}$ (resp. $\mathcal{U}$) when their arguments are x and y. For convenience, we will drop the arguments x, y, $\mathcal{F}$, and $\mathcal{U}$ in the equations to follow. Note that

$$p_\phi = (b_X - a_X)(b_Y - a_Y)$$

and

$$p_\psi = \left(\frac{N_X - M_X}{2^R}\right)\left(\frac{N_Y - M_Y}{2^R}\right).$$

Therefore, $|p_\psi - p_\phi|$ is bounded above by

$$\left|\left(\frac{N_X - M_X}{2^R}\right) - \left(b_X - a_X\right)\right|$$

$$\begin{aligned}
&+ \left|\left(\frac{N_Y - M_Y}{2^R}\right) - \left(b_Y - a_Y\right)\right| \\
\leq\ & \left|\frac{N_X}{2^R} - b_X\right| + \left|\frac{M_X}{2^R} - a_X\right| \\
&+ \left|\frac{N_Y}{2^R} - b_Y\right| + \left|\frac{M_Y}{2^R} - a_Y\right| \\
\leq\ & 4\left(\frac{1}{2}\cdot 2^{-R}\right) \\
=\ & 2\cdot 2^{-\alpha L}.
\end{aligned}$$

Thus,

$$|p_\psi(\mathcal{U}, x, y) - p_\phi(\mathcal{F}, x, y)| \leq 2\cdot 2^{-\alpha L}.$$

But, $\epsilon_\phi(f; x, y)$ (resp. $\epsilon_\psi(f; x, y)$) is just the sum of $p_\phi(\mathcal{F}, x, y)$ (resp. $p_\psi(\mathcal{U}, x, y)$) over all leaves $\mathcal{F}$ (resp. $\mathcal{U}$) whose computed value does not equal $f(x, y)$. From the last inequality, and the fact that the total number of leaves is no more than 2^L, we get

$$\begin{aligned}
\epsilon_\psi(f; x, y) &\leq \epsilon_\phi(f; x, y) + 2^L\left(2\cdot 2^{-\alpha L}\right) \\
&\leq \epsilon + 2\cdot 2^{-(\alpha-1)L}.
\end{aligned}$$

Since this holds for every pair of arguments x and y, we have

$$\epsilon_\psi(f) \leq \epsilon + 2\cdot 2^{-(\alpha-1)L}.$$

This completes the proof. □

From now on, we will forget about ϕ and focus attention on ψ. In fact, it is ψ that we will simulate over the noisy channels.

4 Generating the random inputs for ψ from noise

In order to execute the protocol ψ, each processor needs $R = \lceil \alpha L \rceil$ independent and unbiased bits as a random input. However, our objective is to simulate ψ *deterministically* over a pair of BSCs (each having crossover probability $0 < p < 1/2$). Here is where channel noise actually helps, as a readily available source of randomness! Suppose one processor transmits N 0's across its BSC to the other. The channel output is then an i.i.d. sequence, $Z_1, Z_2, \ldots, Z_N$, of 0-1 valued random variables, each of which is 1 with probability p. In this section, we show how this random sequence can be processed deterministically to generate R independent and unbiased bits, except for a certain failure probability. This probability approaches zero exponentially as R increases, provided the "rate" N/R is maintained at some fixed level above $1/h(p)$. Thus, as far as this part of the simulation is concerned, a noisier channel is better.

Theorem 4.1 *Let* $N = \lceil \beta R/h(p) \rceil$ *(for some* $\beta > 1$*). There exists a mapping* $g : \{0,1\}^N \rightarrow \{\Lambda\} \cup \{0,1\}^R$ *(*Λ *is the empty string), such that, for any* $\mathbf{b} \in \{0,1\}^R$*,*

$$Pr\left\{g\left(Z_1, Z_2, \ldots, Z_N\right) = \mathbf{b}\right\} = \frac{1-\delta}{2^R}$$

where δ*, the probability of generating the empty string, is at most*

$$\min_{\substack{0<s<p \\ h(s)>h(p)/\beta}} \left\{2^{-\beta R \frac{D(s\|p)}{h(p)}} + 2^{-\beta R \frac{D(1-s\|p)}{h(p)}} + (\beta R/h(p))\, 2^{-\frac{R}{h(p)}[\beta h(s) - h(p)]}\right\}.$$

Proof: Choose any s satisfying $0 < s < p$ and $h(s) > h(p)/\beta$. Consider the following map from $\{0,1\}^N$ to $\{\Lambda\} \cup \{0,1\}^R$, based on the type of a sequence:

For $0 \leq k \leq N$:

a) if $h(k/N) < h(s)$, map all sequences having exactly k 1's to Λ.

b) if $h(k/N) \geq h(s)$, partition the $\binom{N}{k}$ sequences having exactly k 1's into $2^R + 1$ disjoint subsets, with each of the first 2^R subsets having exactly $\left\lfloor \binom{N}{k}/2^R \right\rfloor$ elements, and the last one having the remaining $\binom{N}{k} \bmod 2^R$ elements. Then, map each of the first 2^R subsets to one of the 2^R 0-1 sequences of length R in an arbitrary 1-1 way. Map all the sequences in the last subset to Λ.

With this map, it is clear that all 2^R 0-1 sequences of length R are equiprobable (since any two sequences of the same type have the same probability, and, by construction, every $\mathbf{b} \in \{0,1\}^R$ has the same number of preimages in any given type class).

It only remains to prove that δ, the probability of generating the empty string, is bounded above as claimed. This is done using standard large deviation bounds. Let $q = 1 - p$ in what follows.

$$\begin{aligned}
\delta &= \sum_{\substack{0 \leq k \leq N \\ h(k/N) \geq h(s)}} \left[\binom{N}{k} \bmod 2^R \right] p^k q^{N-k} \\
&\quad + \sum_{\substack{0 \leq k \leq N \\ h(k/N) < h(s)}} \binom{N}{k} p^k q^{N-k} \\
&\leq 2^R \sum_{s \leq k/N \leq 1-s} p^k q^{N-k} \\
&\quad + \sum_{k/N < s} \binom{N}{k} p^k q^{N-k} \\
&\quad + \sum_{k/N > 1-s} \binom{N}{k} p^k q^{N-k}.
\end{aligned}$$

The first term equals

$$\begin{aligned}
& 2^R \sum_{s \leq k/N \leq 1-s} 2^{-N[h(k/N) + D(k/N \| p)]} \\
&\leq 2^R 2^{-Nh(s)} \sum_{s \leq k/N \leq 1-s} 2^{-ND(k/N \| p)} \\
&\leq 2^R 2^{-Nh(s)} \sum_{0 < k < N} 1 \\
&= (N-1) 2^R 2^{-Nh(s)} \\
&\leq (\beta R / h(p)) 2^{-\frac{R}{h(p)}[\beta h(s) - h(p)]}.
\end{aligned}$$

By the Chernoff bound for the tails of a binomial distribution

$$\begin{aligned}
\sum_{k/N < s} \binom{N}{k} p^k q^{N-k} &\leq 2^{-N \cdot D(s \| p)} \\
&\leq 2^{-\beta R \frac{D(s \| p)}{h(p)}}
\end{aligned}$$

and

$$\begin{aligned}
\sum_{k/N > 1-s} \binom{N}{k} p^k q^{N-k} &\leq 2^{-N \cdot D(1-s \| p)} \\
&\leq 2^{-\beta R \frac{D(1-s \| p)}{h(p)}}.
\end{aligned}$$

Therefore

$$\begin{aligned}
\delta \leq\ & (\beta R / h(p)) 2^{-\frac{R}{h(p)}[\beta h(s) - h(p)]} \\
& + 2^{-\beta R \frac{D(s \| p)}{h(p)}} + 2^{-\beta R \frac{D(1-s \| p)}{h(p)}}.
\end{aligned}$$

This holds for every s satisfying $0 < s < p$ and $h(s) > h(p)/\beta$. In particular, choose that s which minimizes the above expression, to complete the proof. □

So, before the actual simulation of ψ, each processor sends $N = \lceil \beta R / h(p) \rceil$ 0's across its BSC to the other (here, $R = \lceil \alpha L \rceil$), which then attempts to generate R unbiased bits, by the above procedure. The probability that at least one of them fails to do so is bounded above by twice

$$\min_{\substack{0<s<p \\ h(s)>h(p)/\beta}} \left\{ 2^{-\beta\alpha L \frac{D(s\|p)}{h(p)}} + 2^{-\beta\alpha L \frac{D(1-s\|p)}{h(p)}} \right. \\ \left. + \ (\beta \lceil \alpha L \rceil / h(p))\, 2^{-\frac{\alpha L}{h(p)}[\beta h(s) - h(p)]} \right\}.$$

5 Simulation of ψ over the noisy channels

In this section, we will describe how the actual simulation of ψ over the noisy channels is to be carried out. We will assume, for the purposes of this section, that both A_X and A_Y have successfully generated the random inputs they need to execute ψ. Thus, A_X has an input $(x, \omega_X) \in \mathcal{X}_\psi$, and A_Y has an input $(y, \omega_Y) \in \mathcal{Y}_\psi$.

5.1 Modifications to T_ψ

It will be convenient to modify T_ψ in two ways:

a) Starting from the root and proceeding level by level, check if any node corresponds to an empty rectangle. If so, delete that node and all its descendants from the tree, and collapse its sibling and parent (which must correspond to the same nonempty rectangle) into one node. (This amounts to getting rid of all redundant transmissions in ψ.) All nodes in the new tree correspond to nonempty rectangles.

b) Extend each leaf of the new tree that is a left child (resp. right child) by a sequence of left (resp. right) edges, up to depth $2L$. The rectangle corresponding to any of these new nodes is assumed to be the same as that of the leaf of which it is a descendant. All the new left (resp. right) edges are labelled by 0 (resp. 1). We will refer to the original leaves of T_ψ as *parent-leaves*, and to all the new nodes as *pseudo-leaves*. The term *internal node* will be used only for the internal nodes of the original tree.

To each internal node $\mathcal{U}$, associate a *transmitter* $\pi(\mathcal{U})$ (either X or Y) in the following way: if $\mathcal{U}$ is X-partitioned (resp. Y-partitioned), then $\pi(\mathcal{U})$ equals X (resp. Y). Assume, without loss of generality, that the root is X-partitioned, i.e., that the first transmission on any input is by A_X. If $\mathcal{U}$ is a parent-leaf, define $\pi(\mathcal{U})$ to be the transmitter associated to its parent. If $\mathcal{U}$ is a pseudo-leaf, define $\pi(\mathcal{U})$ and $V_\psi(\mathcal{U})$ to be the transmitter and computed value associated to the unique parent-leaf of which it is a descendant.

Intuitively, these modifications have the effect that, on each input, the processor that transmits last simply repeats its last transmission until there have been $2L$ transmissions in all.

For convenience, we will continue to refer to this modified tree as T_ψ. Note that these modifications cannot increase the complexity or probability of error in computing f.

Let $\mathcal{U} = U_X \times U_Y$ be any node in T_ψ. Define $B_X[\mathcal{U}; x, \omega_X]$ for each $(x, \omega_X) \in U_X$ in the following way:

a) If $\pi(\mathcal{U}) = X$, let $B_X[\mathcal{U}; x, \omega_X]$ be the bit that A_X transmits at $\mathcal{U}$ on the input (x, ω_X).

b) If $\pi(\mathcal{U}) = Y$, let $B_X[\mathcal{U}; x, \omega_X]$ be the dummy symbol $*$.

Define $B_Y[\mathcal{U}; y, \omega_Y]$ analogously, for each $(y, \omega_Y) \in U_Y$.

During the simulation, each processor is given a "pebble" which it moves between the nodes of T_ψ according to certain rules. L, R, B, and H will denote the "pebble moves" of left, right, back, and halt respectively. We will use $(\mathcal{U} : L)$, $(\mathcal{U} : R)$, and $(\mathcal{U} : B)$ to denote the left child, right child, and parent respectively of $\mathcal{U}$. $(\mathcal{U} : H)$ will denote $\mathcal{U}$ itself. Note that not all pebble moves are possible at all nodes.

5.2 Construction of state trees

For the purposes of the simulation, A_X and A_Y construct two different *state trees* Σ_X and Σ_Y respectively, whose construction we describe next. In what follows, Z represents either X or Y:

Σ_Z is a tree of depth $2L$, every leaf of which is at level $2L$ (the root is at level 0). Each internal node in Σ_Z has at most 8 children. Each edge is labelled by a *track* τ, which has two components: the first is a "pebble move" $pebmov(\tau)$ (L, R, H, or B), and the second is a "transmission" $trans(\tau)$ (0, 1, or $*$). A state s at level t in Σ_Z $(1 \leq t \leq 2L)$ will be referred to by the sequence of t tracks labelling the t edges leading to it from the root. (The root itself is represented by the null sequence Λ.) $(pebmov_i(s), trans_i(s))$ will denote the i^{th} of these tracks. (It will be seen that all edges originating from a given state are labelled by distinct tracks, so that there can be no ambiguity in referring to states in this way.) If some edge originating at state s is labelled with the track τ, then $(s : \tau)$ will refer to the corresponding child of s.

Each state s corresponds to a "pebble position" $pebpos(s)$ (a node in T_ψ). The root of Σ_Z corresponds to the root of T_ψ. If $s = (s' : \tau)$, then $pebpos(s) = (pebpos(s') : pebmov(\tau))$.

The children of any state s *in* Σ_Z can be figured out as follows: let M represent any pebble move that is possible at $v = pebpos(s)$. Let $v' = (v : M)$. Then, s has a child corresponding to the track

a) $(M, 0)$ iff $\pi(v') = Z$ and the pebble move L is possible at v'.

b) $(M, 1)$ iff $\pi(v') = Z$ and the pebble move R is possible at v'.

c) $(M, *)$ iff $\pi(v')$ is not Z.

(This is where the difference between Σ_X and Σ_Y comes in.) Note that, if $\pi(pebpos(s)) = Z$, for some state s at level $t \geq 1$ in Σ_Z, then $trans_t(s)$ is 0 or 1. Otherwise $trans_t(s)$ is $*$.

Since there are at most 4 possible pebble moves at $pebpos(s)$, and for each possible move, s has 1 or 2 children, the total number of children of s does not exceed 8 (as claimed earlier). This concludes the description of the state trees.

The following lemma is crucial:

Lemma 5.1 *Let $d \geq 2$ and $0 < \mu < 1$. Let M be any integer such that $M \geq d$, $M \geq 1/(1-\mu)$, and $D(1-\mu \parallel 1/M) >$*

$\log_2 2d^2$. *Then, there exists a labelling of the edges of a complete d-ary tree of any finite depth with the integers* $1, 2, \ldots, M$ *such that the following condition ("relative Hamming distance* $\geq \mu$*") is satisfied:*

For any $l \geq 1$, *and any two paths of length* l *originating at the same node in the tree, the Hamming distance between the sequences of integers labelling the two paths is at least* μl.

Proof: Appendix. □

When $d = 8$ and $\mu = 1/2$, it is easily verified that $M = 2^{16}$ satisfies all the conditions in the lemma. Thus, it is possible to label the edges of Σ_X and Σ_Y with the integers $1, 2, \ldots, 2^{16}$ in such a way that the "relative Hamming distance $\geq 1/2$" condition is satisfied. Fix some such labelling in both Σ_X and Σ_Y. We will refer to these integer labels as *indices.* If s is a state at level $t \geq 1$ (in either state tree), then, for $1 \leq i \leq t$, $\mathcal{I}[s; i]$ will denote the index labelling the i^{th} edge from the root to s. (It will be clear from the context which state tree is being referred to.)

In each round of the simulation, each processor will have to convey an index to the other. For this purpose, A_X and A_Y agree on a common *block code* for the set of 2^{16} indices, of blocklength n. (We will specify later how n is to be chosen.) Any such code is specified by an *encoding map* $C : \{1, 2, \ldots, 2^{16}\} \rightarrow \{0, 1\}^n$ and *decoding sets* $\mathcal{D}_1, \mathcal{D}_2, \ldots, \mathcal{D}_{2^{16}}$ which partition $\{0, 1\}^n$. The worst-case probability of error associated with such a code when it is used over the BSCs connecting the two processors is defined as

$$\max_{1 \leq k \leq 2^{16}} Pr\left\{\overline{\mathcal{D}_k} \middle| C(k)\right\}.$$

For purposes of analysis, we may assume that A_X and A_Y choose that block code of blocklength n (having 2^{16} codewords) which has the *least* worst-case error probability. We will denote this minimum error probability by $\delta(n)$. (An exact formula for $\delta(n)$ is not known. Ultimately, we will upper bound $\delta(n)$ by considering a simple *repetition code.*)

5.3 Simulation algorithm

The simulation of ψ on any pair of inputs, (x, ω_X) and (y, ω_Y), proceeds in $2L$ *rounds*, indexed by $t = 1, 2, \ldots, 2L$. To start with, at $t = 0$, each processor is at the root of its own state tree ($s_X(0) = s_Y(0) = \Lambda$), and has its pebble at the root of T_ψ ($\mathcal{U}_X(0) = \mathcal{U}_Y(0) = \mathcal{X}_\psi \times \mathcal{Y}_\psi$).

In the t^{th} round ($1 \leq t \leq 2L$), A_X does the following:

a) It decides on a *pebble move* for the t^{th} round, $mov_X(t)$ (we will explain later how), and updates its pebble position to

$$\mathcal{U}_X(t) = (\mathcal{U}_X(t-1) : mov_X(t)).$$

b) It takes its *track* for the t^{th} round to be

$$\tau_X(t) = (mov_X(t), B_X[\mathcal{U}_X(t); x, \omega_X])$$

and accordingly updates its state to

$$s_X(t) = (s_X(t-1) : \tau_X(t)).$$

Note that $\mathcal{U}_X(t) = pebpos(s_X(t))$.

c) It then takes its *index* for the t^{th} round to be

$$m_X(t) = \mathcal{I}[s_X(t); t],$$

i.e., the index labelling the edge in Σ_X along which it just moved, and transmits the n-bit codeword for $m_X(t)$ across its BSC to A_Y. Assume that A_Y decodes this message as $\hat{m}_X(t)$.

Similarly, A_Y decides on a pebble move $mov_Y(t)$, track $\tau_Y(t)$, and an index $m_Y(t)$ for the t^{th} round ($1 \leq t \leq 2L$), updates its pebble position and state to $\mathcal{U}_Y(t)$ and $s_Y(t)$ respectively, and transmits the n-bit codeword for $m_Y(t)$ across its BSC to A_X. Assume that A_X decodes this message as $\hat{m}_Y(t)$.

A_X also maintains an *estimate*, $\hat{s}_Y(t)$, of $s_Y(t)$ (for $1 \leq t \leq 2L$), based on the sequence of indices $(\hat{m}_Y(1), \ldots, \hat{m}_Y(t))$ it has received from A_Y in the first t rounds. This estimate is computed using the following (suboptimal) *minimum-distance rule*:

Choose $\hat{s}_Y(t)$ to be that state s at level t in Σ_Y, for which the sequence of indices $(\mathcal{I}[s;1], \mathcal{I}[s;2], \ldots, \mathcal{I}[s;t])$ is closest in Hamming distance to the received sequence $(\hat{m}_Y(1), \hat{m}_Y(2), \ldots, \hat{m}_Y(t))$. Resolve ties arbitrarily.

Similarly, A_Y maintains an estimate, $\hat{s}_X(t)$, of $s_X(t)$ (for $1 \leq t \leq 2L$), based on $(\hat{m}_X(1), \ldots, \hat{m}_X(t))$.

Finally, define $\hat{s}_X(0)$ and $\hat{s}_Y(0)$ to be the roots of the respective state trees.

In round t (for $1 \leq t \leq 2L$), A_X uses the state estimate $\hat{s}_Y(t-1)$ to determine its pebble move $mov_X(t)$ in the following way:

a) If $\mathcal{U}_X(t-1) = pebpos\,[\hat{s}_Y(t-1)]$, then A_X assumes that the two pebbles indeed coincide, and decides to simulate a step in the protocol ψ. To do this, it checks who transmits at the node $\mathcal{U}_X(t-1)$. If it is A_X, then it takes $mov_X(t)$ to be L or R, according as $B_X\,[\mathcal{U}_X(t-1); x, \omega_X] = 0$ or 1. If it is A_Y, then it takes $mov_X(t)$ to be L or R, according as $trans_{t-1}\,(\hat{s}_Y(t-1)) = 0$ or 1.

If $\mathcal{U}_X(t-1) \neq pebpos\,[\hat{s}_Y(t-1)]$, then A_X decides to take remedial action to bring the two pebbles together. There are two cases here:

b) If $\mathcal{U}_X(t-1)$ is a strict ancestor of $pebpos\,[\hat{s}_Y(t-1)]$, then A_X takes $mov_X(t)$ to be H, in the hope that A_Y will move its pebble back until the two pebbles coincide.

c) Otherwise, A_X takes $mov_X(t)$ to be B, hoping that A_Y will either stay put at its current pebble position or move its pebble back (as appropriate), until they meet at the least common ancestor.

Similarly, A_Y uses the state estimate $\hat{s}_X(t-1)$ to determine its own pebble move $mov_Y(t)$ in round t, for $2 \leq t \leq 2L$. In the first round, A_Y has no information to base its pebble move on, and therefore always takes $mov_Y(1) = H$.

The total number of transmissions used by both the processors is $4nL$. At the end of $2L$ rounds, A_X checks if $\mathcal{U}_X(2L)$ is a (parent- or pseudo-) leaf of T_ψ. If so, it takes the output of the simulation to be $V_\psi(\mathcal{U}_X(2L)$, the computed value associated with that leaf. If $\mathcal{U}_X(2L)$ is not a leaf, then A_X concludes that the simulation failed. A_Y makes a similar decision based on $\mathcal{U}_Y(2L)$.

We will show that, with high probability, both processors will arrive at the correct computed value of ψ for the given pair of inputs (x, ω_X) and (y, ω_Y).

5.4 Analysis

We will use $lca(\cdot,\cdot)$ to denote the least common ancestor of a pair of nodes in T_ψ, or a pair of states in Σ_X or Σ_Y.

Definition 5.1 *For $0 \leq t \leq 2L$, $mark(t)$ equals*

$$2\{depth\,[lca(\mathcal{U}_X(t), \mathcal{U}_Y(t))]\} - \max\{depth(\mathcal{U}_X(t), depth(\mathcal{U}_Y(t))\}.$$

Observe that $mark(0)$ equals 0 and $mark(1)$ equals -1. In general, $mark(t)$ is an indication of the progress of the simulation at the end of the t^{th} round. From the description of the algorithm, it should be clear that, for any t, the X-projection (resp. Y-projection) of the least common ancestor of $\mathcal{U}_X(t)$ and $\mathcal{U}_Y(t)$ contains the input (x, ω_X) (resp. (y, ω_Y)). So, while either processor's pebble may deviate from the correct path in T_ψ, their least common ancestor is always on the right path. Consequently, if $lca(\mathcal{U}_X(2L), \mathcal{U}_Y(2L))$ is a (parent- or pseudo-) leaf of T_ψ, the simulation must be successful.

Lemma 5.2 *The simulation is successful if $mark(2L) \geq L$.*

Proof: $mark(2L) \geq L$ implies that

$$depth\,[lca(\mathcal{U}_X(2L), \mathcal{U}_Y(2L))] \geq L$$

which means that $lca(\mathcal{U}_X(2L), \mathcal{U}_Y(2L))$ is a leaf of T_ψ. □

Definition 5.2 *For $1 \leq t \leq 2L$, the t^{th} round is "good" for A_X (resp. A_Y) if $\hat{s}_Y(t) = s_Y(t)$ (resp. $\hat{s}_X(t) = s_X(t)$). The t^{th} round is "good" if it is good for both A_X and A_Y.*

Lemma 5.3 *For $1 \leq t \leq 2L - 1$, if the t^{th} round is good, then*

$$mark(t+1) = mark(t) + 1.$$

If the t^{th} round is bad, then

$$mark(t+1) \geq mark(t) - 3.$$

Proof: For both parts, consider three cases: (a) $\mathcal{U}_X(t) = \mathcal{U}_Y(t)$ (b) one of $\mathcal{U}_X(t)$ and $\mathcal{U}_Y(t)$ is a strict ancestor of the other (c) the least common ancestor of $\mathcal{U}_X(t)$ and $\mathcal{U}_Y(t)$ is not equal to either $\mathcal{U}_X(t)$ or $\mathcal{U}_Y(t)$. The analysis is straightforward. □

Corollary 5.1 *The simulation is successful if the number of bad rounds, out of rounds $1, 2, \ldots, 2L-1$, is less than $(L-1)/4$.*

Proof: If the number of bad rounds is less than $(L-1)/4$, then $mark(2L) - mark(1)$ is

$$\begin{aligned} &> 1 \cdot \left(2L - 1 - \frac{L-1}{4}\right) - 3 \cdot \left(\frac{L-1}{4}\right) \\ &= L. \end{aligned}$$

Since $mark(1) = -1$, this means that $mark(2L) \geq L$. □

Assume $L \geq 2$, so that $(L-1)/4 \geq L/8$. The probability of failure is then bounded above by the probability of having at least $L/8$ bad rounds (out of rounds

$1, 2, \ldots, 2L-1$), which is no greater than

$$\begin{aligned} & Pr\{A_X \text{ has } \geq L/16 \text{ bad rounds}\} \\ + \ & Pr\{A_Y \text{ has } \geq L/16 \text{ bad rounds}\} \end{aligned}$$

since a round is bad if and only if it is bad for at least one processor. We will now estimate the first of these probabilities. A similar analysis will hold for the second one.

Definition 5.3 *Suppose the t^{th} round is bad for A_X. Then, the* **error interval** *associated with this bad round is the sequence of rounds $t-l+1, t-l+2, \ldots, t$. Here, l is the* **length** *of the error interval, defined as $t - depth\,[lca\,(\hat{s}_Y(t), s_Y(t))]$.*

Now, the number of bad rounds for A_X is clearly no greater than the size of the union of the error intervals associated with all the bad rounds (since each bad round is contained in the error interval it defines). Therefore,

$$\begin{aligned} & Pr\{A_X \text{ has } \geq L/16 \text{ bad rounds}\} \\ \leq \ & Pr\{\text{union of all the error intervals} \\ & \qquad \text{for } A_X \text{ has size} \geq L/16\}. \end{aligned}$$

Lemma 5.4 *If the size of the union of all error intervals for A_X is $\geq L/16$, then there must exist* **disjoint** *error intervals, the sum of whose lengths is $\geq L/32$.*

Proof: Omitted (see [3]). □

Consequently, it suffices to estimate the probability that A_X has certain bad rounds defining disjoint error intervals of total length $\geq L/32$.

Lemma 5.5 *The probability of A_X having bad rounds that define disjoint error intervals of total length $\geq L/32$ is no greater than*

$$2^{4L}\left(8 \cdot [\delta(n)]^{1/4}\right)^{L/32}$$

provided $8 \cdot [\delta(n)]^{1/4} < 1$.

Proof: Let rounds $t_1, t_2, \ldots, t_k$ be bad rounds for A_X, defining disjoint error intervals of lengths $l_1, l_2, \ldots, l_k$ (for some $1 \leq k \leq 2L-1$, $l_1, l_2, \ldots, l_k \geq 1$, $0 \leq t_1 - l_1 < t_1 \leq t_2 - l_2 < \cdots \leq t_k - l_k < t_k \leq 2L-1$). Also, assume that $l_1 + \cdots + l_k \geq L/32$.

For each choice of $k, t_1, \ldots, t_k, l_1, \ldots, l_k$, there are several ways of choosing "wrong" states $s_1, s_2, \ldots, s_k$ at levels $t_1, t_2, \ldots, t_k$, so as to get error intervals of lengths $l_1, l_2, \ldots, l_k$. This number is

$$\begin{aligned} \leq \ & (8^{l_1}-1)(8^{l_2}-1)\cdots(8^{l_k}-1) \\ < \ & 8^{(l_1+\cdots+l_k)} \end{aligned}$$

since each state has at most 8 children. Fix some such choice of $s_1, \ldots, s_k$.

Let us bound $p_e(s_1, \ldots, s_k)$, the probability that A_X takes $\hat{s}_Y(t_i)$ to be s_i (for each $1 \leq i \leq k$). To do this, first condition on the final state of A_Y, viz., $s_Y(2L)$. Let $p_e(s_1, \ldots, s_k|s)$ be the probability that A_X takes $\hat{s}_Y(t_i)$ to be s_i (for each $1 \leq i \leq k$), *given that $s_Y(2L)$ equals s* (for some state s at level $2L$ in Σ_Y).

Let $m(t) = \mathcal{I}[s;t]$ $(1 \leq t \leq 2L)$. These are the actual indices transmitted by A_Y during the simulation, if its final state is s. Assume that A_X decode these as $\hat{m}(1), \hat{m}(2), \ldots, \hat{m}(2L)$. And, for $1 \leq i \leq k$ and $1 \leq t \leq t_i$, let $q^{(i)}(t) = \mathcal{I}[s_i; t]$. These are the indices labelling the edges from the root to s_i. Finally, let $M_{t,t'}$ denote the sequence $(m(t), m(t+1), \ldots, m(t'))$. Similarly, define $\hat{M}_{t,t'}$ and $Q^{(i)}_{t,t'}$.

Note that $p_e(s_1, \ldots, s_k | s)$ is bounded above by

$$Pr\Big\{ d[\hat{M}_{1,t_i}, Q^{(i)}_{1,t_i}] \leq d[\hat{M}_{1,t_i}, M_{1,t_i}] \text{ for } i = 1, 2, \ldots, k \,\Big|\, s_Y(2L) = s \Big\}$$

since $\hat{s}_Y(t_i) = s_i$ means that $\hat{m}(1, t_i)$ is closest to $q_i(1, t_i)$ (by the minimum-distance rule).

But $M_{1,t_i - l_i} = Q^{(i)}_{1,t_i - l_i}$, since, by assumption, the length of the error interval corresponding to round t_i is l_i. Therefore, the above probability is equal to

$$Pr\Big\{ d[\hat{M}_{t_i - l_i + 1, t_i}, Q^{(i)}_{t_i - l_i + 1, t_i}] \leq d[\hat{M}_{t_i - l_i + 1, t_i}, M_{t_i - l_i + 1, t_i}] \text{ for } i = 1, 2, \ldots, k \,\Big|\, s_Y(2L) = s \Big\}.$$

Further, by the "relative Hamming distance $\geq 1/2$" condition,

$$d[M_{t_i - l_i + 1, t_i}, Q^{(i)}_{t_i - l_i + 1, t_i}] \geq l_i/2.$$

This, together with the triangle inequality, means that if

$$d[\hat{M}_{t_i - l_i + 1, t_i}, Q^{(i)}_{t_i - l_i + 1, t_i}] \leq d[\hat{M}_{t_i - l_i + 1, t_i}, M_{t_i - l_i + 1, t_i}]$$

then

$$d[\hat{M}_{t_i - l_i + 1, t_i}, M_{t_i - l_i + 1, t_i}] \geq l_i/4.$$

Consequently, the last probability is no greater than

$$Pr\Big\{ d[\hat{M}_{t_i - l_i + 1, t_i}, M_{t_i - l_i + 1, t_i} \geq l_i/4] \text{ for } i = 1, 2, \ldots, k \,\Big|\, s_Y(2L) = s \Big\}.$$

By the disjointness of the error intervals and the memorylessness of the channel, this is the probability of the intersection of k *independent* events, and is therefore equal to

$$\begin{aligned} & \prod_{i=1}^{k} Pr\Big\{ d[\hat{M}_{t_i - l_i + 1, t_i}, M_{t_i - l_i + 1, t_i} \geq l_i/4] \text{ for } i = 1, 2, \ldots, k \,\Big|\, s_Y(2L) = s \Big\} \\ \leq & \prod_{i=1}^{k} [\delta(n)]^{l_i/4} \\ = & [\delta(n)]^{\frac{l_1 + \cdots + l_k}{4}} \end{aligned}$$

since the probability of error for any index is at most $\delta(n)$. So

$$p_e(s_1, \ldots, s_k | s) \leq [\delta(n)]^{\frac{l_1 + \cdots + l_k}{4}}.$$

Since this holds for any s, we can remove the conditioning on $s_Y(2L)$, to get

$$p_e(s_1, \ldots, s_k) \leq [\delta(n)]^{\frac{l_1 + \cdots + l_k}{4}}.$$

Now, take the union over all possible choices of $s_1, \ldots, s_k$ (no more than $8^{(l_1 + \cdots + l_k)}$ in number) to conclude that the probability that rounds $t_1, \ldots, t_k$ are bad for A_X and have associated error intervals of lengths $l_1, \ldots, l_k$ is

$$\begin{aligned} \leq & \; 8^{(l_1 + \cdots + l_k)} \cdot [\delta(n)]^{\frac{l_1 + \cdots + l_k}{4}} \\ = & \left(8 \, [\delta(n)]^{1/4}\right)^{(l_1 + \cdots + l_k)} \\ \leq & \left(8 \, [\delta(n)]^{1/4}\right)^{L/32} \end{aligned}$$

provided $8\,[\delta(n)]^{1/4} < 1$. (Recall that $l_1 + \cdots + l_k$ was assumed to be at least $L/32$.)

Finally, there are no more than $2^{2L} \cdot 2^{2L}$ ways of choosing $k, t_1, \ldots, t_k, l_1, \ldots, l_k$. Taking the union over all such choices, we can conclude that the probability of A_X having bad rounds that define disjoint error intervals of total length $\geq L/32$ is

$$\leq 2^{4L}\left(8\,[\delta(n)]^{1/4}\right)^{L/32}$$

as claimed. □

It follows that the probability that A_X has at least $L/16$ bad rounds is bounded above by the same quantity. A similar analysis holds for A_Y also. Putting together all the pieces now, we can conclude that the probability that the simulation fails on any pair of inputs is

$$\leq 2 \cdot 2^{4L}\left(8\,[\delta(n)]^{1/4}\right)^{L/32}$$

provided $8\,[\delta(n)]^{1/4} < 1$.

An exact formula for $\delta(n)$ is not known. We can, however, upper bound it by considering the simplest possible code, viz., a *repetition code.* Represent each index by 16 bits, and encode each bit by simply repeating it r times (for a total of $n = 16r$ bits). Use *majority decoding.* The probability that a bit is decoded wrongly is no greater than $2^{-r \cdot D(\frac{1}{2}\|p)}$ (Chernoff bound). A union bound now gives

$$\delta(n) \leq 16 \cdot 2^{-n \cdot D(\frac{1}{2}\|p)/16}.$$

Since $D(\frac{1}{2}\|p) > 1 - h(p)$ for $0 < p < 1/2$, we have

$$\delta(n) \leq 16 \cdot 2^{-\frac{n(1-h(p))}{16}}.$$

Therefore

$$2 \cdot 2^{4L}\left(8\,[\delta(n)]^{1/4}\right)^{L/32} \leq 2 \cdot 2^{-\left(\frac{n(1-h(p))-8448}{2048}\right)L}.$$

This bound is meaningful only if $n = \lceil \gamma/(1-h(p)) \rceil$, for some $\gamma > 8448$. In this case, it can be verified that the condition $8\,[\delta(n)]^{1/4} < 1$ is also satisfied. We have proved the following result:

Theorem 5.1 *Let* $n = \lceil \gamma/(1-h(p)) \rceil$ *(for some* $\gamma > 8448$*). It is possible for the two processors to simulate* ψ *over the BSCs on any pair of inputs, using* $4nL$ *transmissions in all, with a failure probability (in arriving at the correct computed value of* ψ*) of no more than* $2 \cdot 2^{-\left(\frac{\gamma-8448}{2048}\right)L}$.

We have now completed the proof of the main theorem stated in the introduction.

Appendix

Proof of Lemma 5.1: Fix any M satisfying the conditions in the statement, viz., $M \geq d$, $M \geq 1/(1-\mu)$, and $D(1-\mu \,\|\, 1/M) > \log_2 2d^2$. The proof is by induction on n, the depth of the d-ary tree.

The case $n = 1$ is trivial (since $M \geq d$). Assume that there exists a labelling of the edges of a complete d-ary tree of depth $n-1$ (for some $n \geq 2$) with the integers $1, 2, \ldots, M$, such that the "relative Hamming distance $\geq \mu$" condition is satisfied.

Choose independent random permutations, σ_{ij}, of $\{1, 2, \ldots, M\}$ (for $1 \leq i \leq d$ and $0 \leq j \leq n-1$). Take d copies of the "good" labelled tree of depth $n-1$. In the

i^{th} of these ($1 \leq i \leq d$), replace each integer labelling an edge out of a vertex at level j ($1 \leq j \leq n-1$) by its image under σ_{ij}. (The root is at level 1, its children are at level 2, etc..) Clearly, each of the resulting labelled trees is also "good." Next, join the roots of the d copies to a common vertex, to get a d-ary tree of depth n. (The new vertex is the root of the new tree.) Label the i^{th} edge out of this root with $\sigma_{i0}(1)$.

Our aim is to show that the resulting random labelling satisfies the "relative Hamming distance $\geq \mu$" condition with positive probability (thus establishing the existence of a "good" labelled tree of depth n).

As noted before, the condition is satisfied for any two distinct paths that originate from any node other than the root. So, consider two paths of length k ($1 \leq k \leq n$) originating from the root. Then, we essentially have two independently chosen random strings of integers between 1 and M, labelling the edges along the two paths. The probability that the Hamming distance between these two strings is $< \mu k$ is $\geq 2^{-k \cdot D(\mu \| 1-\frac{1}{M})} = 2^{-k \cdot D(1-\mu \| \frac{1}{M})}$, by the Chernoff bound, provided $\mu < 1 - 1/M$, which we have assumed to be true.

There are no more than $d^k \cdot d^k$ pairs of distinct paths of length k originating from the root. This, together with the union bound, means that the probability that the "relative Hamming distance $\geq \mu$" condition is violated by some pair of paths is

$$\leq \sum_{k=1}^{n} d^{2k} 2^{-k \cdot D(1-\mu \| \frac{1}{M})}$$

$$= \sum_{k=1}^{n} \left[d^2 2^{-D(1-\mu \| \frac{1}{M})} \right]^k.$$

But $D(1-\mu \| 1/M) > \log_2 2d^2$ by assumption, and therefore,

$$d^2 2^{-D(1-\mu \| \frac{1}{M})} < \frac{1}{2}.$$

Consequently,

$$\begin{aligned} &\sum_{k=1}^{n} \left[d^2 2^{-D(1-\mu \| \frac{1}{M})} \right]^k \\ &\quad \leq \frac{d^2 2^{-D(1-\mu \| \frac{1}{M})}}{1 - d^2 2^{-D(1-\mu \| \frac{1}{M})}} \\ &\quad < 1. \end{aligned}$$

This proves the existence of at least one "good" labelling. □

References

[1] L. Lovász. "Communication complexity: A survey". In B.H. Korte et al., editors, *Paths, Flows and VLSI layout.* Springer-Verlag, 1990.

[2] A. Orlitsky and A. El Gamal. "Communication complexity". In Y. Abu-Mostafa, editor, *Complexity in Information Theory.* Springer-Verlag, 1988.

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

[4] A.C. Yao. "Some complexity questions related to distributive computing". *Proc. of the 11th Annual Symposium on the Theory of Computing,* 1979.