Towards Coding for Maximum Errors in Interactive Communication

Mark Braverman* Anup Rao†

Abstract

We show that it is possible to encode any communication protocol between two parties so that the protocol succeeds even if a $(1/4 - \epsilon)$ fraction of all symbols transmitted by the parties are corrupted adversarially, at a cost of increasing the communication in the protocol by a constant factor (the constant depends on epsilon). This encoding uses a constant sized alphabet. This improves on an earlier result of Schulman, who showed how to recover when the fraction of errors is bounded by 1/240. We also show how to simulate an arbitrary protocol with a protocol using the binary alphabet, a constant factor increase in communication and tolerating a $1/8 - \epsilon$ fraction of errors.

1 Introduction

Suppose a sender wants to send an n bit message to a receiver, but some of the sender's transmissions may be received incorrectly. What is the best way to encode the message in order to recover from the errors? This question, first considered by Shannon [Sha48], initiated the study of error correcting codes, which have since found applications in many different contexts. The book [PWJ72] is a good reference. In our work, we study the analogous question in the more general setting of interactive communication. We refer the reader to the book [KN97] for an introduction to interactive communication protocols. Most models of computation, for example circuits, branching programs, streaming algorithms, distributed systems, inherently involve communication protocols, so the following question is well motivated — What is the best way to encode an arbitrary two party communication protocol, so that the protocol cannot be disrupted by errors in the communication? This more general problem was first considered by Schulman [Sch96].

There are several ways to model how errors may occur. Throughout this paper, we focus on the so called *worst-case* scenario; we only assume that the fraction of transmissions that are received incorrectly is bounded, and make no assumption about the distribution of errors[1]. For the case of one way communication considered by Shannon, it is known how to construct error correcting codes that can encode any n-bit message using $O_\epsilon(n)$ bits, so that even if a $(1/4 - \epsilon)$ fraction of the transmitted bits are received incorrectly, the intended message can be recovered. If we allow the transmitted message to use symbols from an alphabet whose size depends on ϵ but remains independent of n, it is known how to encode the message so that it can be recovered even if a $(1/2 - \epsilon)$ fraction of all symbols are corrupted. Moreover, both the encoding and decoding can be implemented by efficient algorithms. It is impossible to recover from an error rate of 1/2 if one wants to recover the intended message exactly, since there can be no valid decoding of a received string that is equal to each of two valid messages half the time.

In a general interactive protocol, each party must wait for a response from the other party before deciding on what to send in each round of communication, so it is not clear how to use error correcting codes to encode the communication of the protocol. If there are R rounds of communication, any encoding of the protocol that works by encoding each round of communication separately can be disrupted with an error rate of $1/2R$ — the errors can completely corrupt the shortest message in the protocol. Thus, this approach is not useful if R is

*University of Toronto. Email: mbraverm@cs.toronto.edu.

†University of Washington, anuprao@cs.washington.edu. Supported by the National Science Foundation under agreement CCF-1016565.

[1] We do assume that transmissions are received in the order they are sent, and that no transmission is dropped.

large. Nevertheless, Schulman showed that any protocol with T bits of communication can be encoded to give a protocol that uses a constant sized alphabet, communicates at most $O(T)$ symbols, and tolerates an error rate of 1/240.

1.1 Our Results

The main result of our work is a new way to encode protocols to tolerate a large fraction of errors. Given a two party communication protocol π, taking inputs from some finite set, we write $\pi(x, y)$ to denote the transcript, namely the sequence of messages exchanged, when π is run with inputs x, y. We shall consider communication protocols that are allowed to send symbols from large alphabets. We write $|\pi|$ for the maximum number of symbols that can be exchanged on any setting of inputs, namely the communication complexity of π. Of course every communication protocol with a larger alphabet can be simulated by a protocol with a binary alphabet, but this translation can change the error rate that a protocol is resilient to.

Our first result uses a non-binary alphabet:

Theorem 1. *For every ϵ, there is an alphabet Σ of size $O_\epsilon(1)$ and a transformation of communication protocols C_ϵ, such that for every binary protocol π, $C_\epsilon(\pi)$ is a protocol using symbols from Σ, $|C_\epsilon(\pi)| = O_\epsilon(|\pi|)$, and for all inputs x, y, both parties can determine $\pi(x, y)$ by running $C_\epsilon(\pi)$ if the fraction of errors is at most $1/4 - \epsilon$.*

In Section 6, we discuss an impossibility result for decoding from larger error rates. We consider the problem where two parties are given bits x, y as inputs and want to compute the parity of the bits. We show that any protocol for this problem that ensures that at each step the parties have a consensus about whose turn it is to transmit (despite arbitrary errors) can be disrupted with an error rate of 1/4. There remains the possibility that the protocol only guarantees a consensus when the error rate is small (say $1/2 - \epsilon$). For this case, to the best of our knowledge it is possible that some encoding can tolerate an error rate as high as $1/2 - \epsilon$.

One can encode/decode every alphabet symbol in the protocol of Theorem 1 using an appropriate binary error correcting code to get a protocol that uses the binary alphabet. The resulting protocol would be able to recover from an error rate of $(1/16 - \epsilon)$. Our second theorem shows how to do better. We can encode the initial protocol into a protocol using a binary alphabet and tolerate $1/8 - \epsilon$ errors.

Theorem 2. *For every ϵ, there is a transformation of communication protocols C_ϵ, such that for every binary protocol π, $C_\epsilon(\pi)$ is a binary protocol, $|C_\epsilon(\pi)| = O_\epsilon(|\pi|)$, and for all inputs x, y, both parties can determine $\pi(x, y)$ by running $C_\epsilon(\pi)$ if the fraction of errors is at most $1/8 - \epsilon$.*

The bottleneck for proving that our encoding scheme is computationally efficient is the efficiency of encoding and decoding tree codes (defined in Section 3). Indeed, if there are polynomial time algorithms for those tasks, then given an algorithm or circuit implementing π, one can very efficiently compute an algorithm or circuit implementing $C_\epsilon(\pi)$.

The rest of the document is organized as follows. In Section 2 we introduce communication protocols and the pointer jumping problem. In Section 3 we introduce tree codes. In Sections 4 and 5 we discuss how to encode arbitrary communication protocols to tolerate errors using a constant sized alphabet, proving Theorem 1. In Section 7 we discuss how to get a result for binary alphabet proving Theorem 2. In Section 6 we describe the argument to show how one cannot tolerate a 1/4 error rate if the parties are always guaranteed to have consensus about whose turn it is to speak.

2 Communication protocols, pointer jumping and errors

Given inputs x, y from some domain, a deterministic protocol with alphabet Σ is a rooted tree where every internal vertex has $|\Sigma|$ children, each corresponding to an element of the alphabet. Every non-leaf vertex in the tree v is associated with a function $f_v(x)$ (or $f_v(y)$) that maps one of the inputs to an element of Σ. The outcome of the protocol on inputs x, y is the unique leaf that is reached by first evaluating the function associated with the root, then evaluating the function associated with the child obtained by the first evaluation, and so on. The depth of the tree, T is called the communication complexity of the protocol. Two parties who each have access to just one of the inputs can compute the outcome of the protocol by communicating at most T symbols to each other, in the obvious way.

In this paper, we only consider deterministic communication protocols, since our results easily extend to the case of randomized protocols by viewing a randomized protocol as a distribution over deterministic protocols.

Let $\mathcal{T}$ be a rooted binary tree of depth T. Let $\mathcal{X}$ denote the set of edges leaving vertices at even depth, and $\mathcal{Y}$ denote the set of edges leaving vertices at odd depth. Given any set A of edges in the tree, we say that A is *consistent* if A contains at most one edge leaving every vertex of the tree. We write $v(A)$ to denote the unique vertex of maximal depth that is reachable from the root using the edges of A.

Let $X \subset \mathcal{X}$, $Y \subset \mathcal{Y}$ be consistent subsets. Then observe that $X \cup Y$ is also consistent. In the pointer jumping problem, the two parties are given such sets X, Y and the goal is compute $v(X \cup Y)$. Since every communication protocol with communication complexity t bits can be reduced to solving pointer jumping on a tree of depth $2t$, it will suffice for us to find a protocol that can compute $v(X \cup Y)$ even if there are transmission errors.

In this paper, we define communication protocols that are resilient to transmission errors. In our protocols, every vertex at odd depth will be labeled by a function of the first party's input, and every vertex at even depth will be labeled by a function of the second party's input[2]. Given such a protocol, each party runs it as above, using her transmissions and the messages she receives to determine her current position in the protocol tree. In general, each party will conclude the protocol at a different leaf.

3 Tree Codes

Given a set A, let A^k represent the k-fold Cartesian product $A \times A \times \ldots \times A$, and $A^{\leq n}$ denote the set $\cup_{k=1}^{n} A^k$. A d-ary *tree code* of depth n and alphabet Σ is a rooted d-ary tree of depth n whose edges are labeled with elements of Σ. The tree defines an encoding function $C : [d]^{\leq n} \to \Sigma$ mapping each vertex in the tree to the label of the final edge on the path leading to the vertex. Define $\overline{C}(v_1, v_2, \ldots, v_k)$ to be the concatenation $C(v_1), C(v_1, v_2), \ldots, C(v_1, \ldots, v_k)$. This is just the labels along the path from the root to the vertex $v = v_1, \ldots, v_k$.

Given distinct vertices $u, v \in [d]^k$ at depth k in the tree, we write $\ell(u, v)$ to denote the quantity $k + 1 - \min_j\{u_j \neq v_j\}$, namely the distance to the closest common ancestor in the tree. The tree code has distance α if for any $k \in \{1, \ldots, n\}$ and any distinct $u, v \in [d]^k$, $\overline{C}(u_1, u_2, \ldots, u_k)$ and $\overline{C}(v_1, v_2, \ldots, v_k)$ differ in at least $\alpha \cdot \ell(u, v)$ coordinates.

The following theorem was proved in [Sch96]:

Theorem 3. *For every $0 < \alpha < 1$ there exists a d-ary tree code of depth n and distance α, using an alphabet of size $d^{O_\alpha(1)}$.*

Given an arbitrary string $z \in \Sigma^k$, we denote by $D(z)$ the vertex $v \in [d]^k$ that minimizes the hamming distance between $\overline{C}(v_1, \ldots, v_k)$ and z.

4 Recovering from errors using a polynomial size alphabet

As a warmup for our construction, we start by showing how to solve the pointer jumping problem even if $(1/4 - \epsilon)$ fraction of the transmissions are corrupted, using an alphabet of size polynomial in the depth of $\mathcal{T}$. Set $R = \lceil T/\epsilon \rceil$. Our protocol will be defined over the alphabet

$$\Pi = \{0, 1, \ldots, R\} \times \{0, 1\}^{\leq 2},$$

[2] We discuss more general types of error correcting protocols, where the turns need not be alternating, in Section 6.

encoded using a tree code. Set $d = |\Pi|$ and let $C : \Pi^{\leq R} \rightarrow \Sigma^{\leq R}$ and $D : \Sigma^{\leq R} \rightarrow \Pi^{\leq R}$ be the encoding and decoding functions of a d-ary tree code of distance $1 - \epsilon$.

Given any $a \in \Pi^k$, a represents a set of at most k edges in the binary tree $\mathcal{T}$, computed as follows. Let $a_i = (r_i, s_i)$, where $r_i \in \{0, 1, \ldots, R\}$ and $s_i \in \{0, 1\}^{\leq 2}$. We think of r_i as a pointer to the r_i'th edge determined by $a_1, \ldots, a_{r_i}$, and of s_i as specifying an edge that is a descendant of the r_i'th edge.

Formally, for $i = 1, \ldots, k$, we use the following procedure to determine an edge e_i from a:

1. If $r_i = 0$, set e'_i to be the edge of depth at most 2 that is specified by the bits s_i.
2. If $r_i < i$ and e_{r_i} has been defined, let e'_i be the edge specified by starting at the child vertex of the edge e_{r_i} and then taking (at most) two steps in the tree using the bits s_i.
3. If e'_i has been assigned an edge, and there there is no $j < i$ for which e_j, e'_i share a common parent, set $e_i = e'_i$. Else leave e_i undefined.

Given such a string, we write $E(a)$ to denote the set of edges that been assigned to e_i for some i in the process above. Observe that for every edge in $E(a)$, there is a unique index i to which the edge has been assigned. Further, $E(a)$ is consistent and $E(a_1, \ldots, a_i) \subseteq E(a_1, \ldots, a_{i+1})$.

Our protocol will proceed in rounds. In the i'th round, the first party (resp. second party) will compute a symbol a_i (resp. b_i) and transmit an element of Σ which is then received as a'_i (resp. b'_i). The transmissions will ensure that $E(a_1, \ldots, a_{i-1}) \subseteq X$ and $E(b_1, \ldots, b_{i-1}) \subseteq Y$. Define

$$A_i = \begin{cases} \emptyset & \text{if } i = 1, \\ E(a_1, \ldots, a_{i-1}) \cup (E(D(b'_1, \ldots, b'_{i-1})) \cap \mathcal{Y}) & \text{else.} \end{cases}$$

and

$$B_i = \begin{cases} \emptyset & \text{if } i = 1, \\ E(b_1, \ldots, b_{i-1}) \cup (E(D(a'_1, \ldots, a'_{i-1})) \cap \mathcal{X}) & \text{else.} \end{cases}$$

Observe that A_i can always be computed by the first party in round i, and B_i can always be computed by the second party in round i. Note that A_i and B_i are both consistent.

4.1 Protocol

A sample run of the protocol is illustrated on Figure 1.

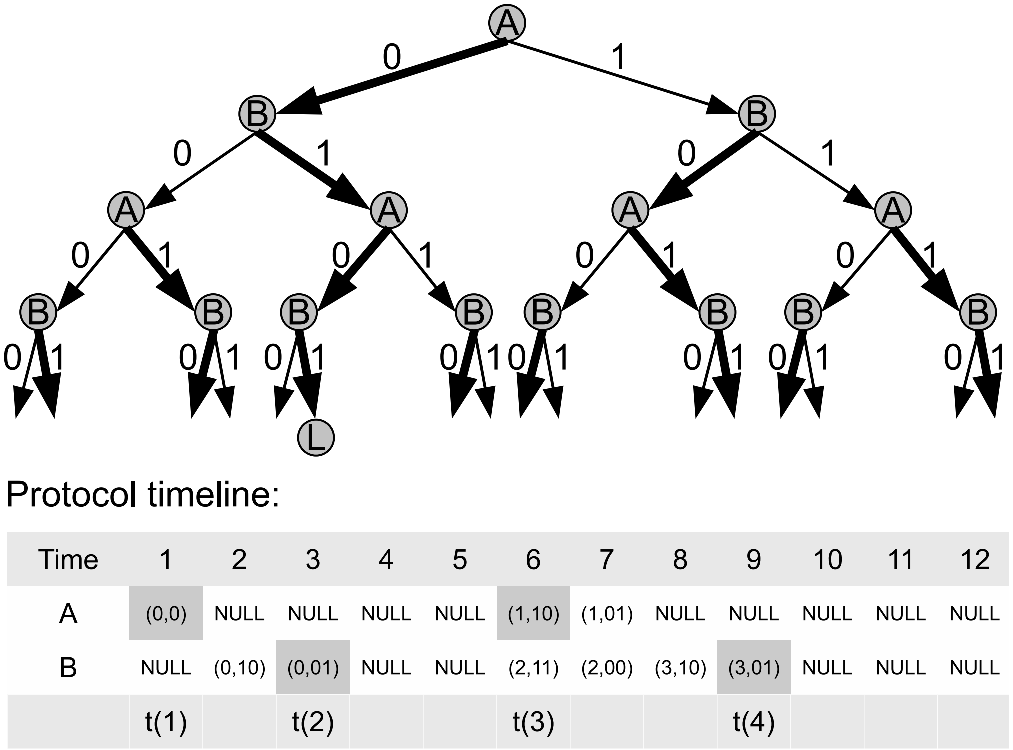
1. For $i = 1, \ldots, R$,

   First Party If $v(A_i) \neq v(A_i \cup X)$, set a_i so that $E(a_1, \ldots, a_i)$ contains the edge out of $v(A_i)$ from the set X. Else, set $a_i = (R, 00)$, so that a_i cannot represent any edge. Transmit $C(a_1, \ldots, a_i)$.

   Second Party If $v(B_i) \neq v(B_i \cup Y)$ set b_i so that $E(b_1, \ldots, b_i)$ contains the edge out of $v(B_i)$ from the set Y. Else, set $b_i = (R, 00)$, so that b_i cannot represent any edge. Transmit $C(b_1, \ldots, b_i)$.

2. The outcome of the protocol from the first (resp. second) party's point of view is the vertex $v(A_{R+1})$ (resp. $v(B_{R+1})$).

Note that the protocol is well defined since if $v(A_i) \neq v(A_i \cup X)$, then $v(A_i)$ must be at even depth, and $E(a_1, \ldots, a_{i-1})$ must contain an edge touching a grandparent of $v(A_i)$ that was transmitted by the first party. Thus there is always a symbol a_i that can extend $a_1, \ldots, a_{i-i}$ to encode the next edge on the path to $v(A_i)$.



Protocol timeline:

Time	1	2	3	4	5	6	7	8	9	10	11	12
A	(0,0)	NULL	NULL	NULL	NULL	(1,10)	(1,01)	NULL	NULL	NULL	NULL	NULL
B	NULL	(0,10)	(0,01)	NULL	NULL	(2,11)	(2,00)	(3,10)	(3,01)	NULL	NULL	NULL
	t(1)		t(2)			t(3)			t(4)			

Figure 1: An illustration of the protocol for recovering from errors using a linear size alphabet. On the top is the tree $\mathcal{T}$ with the edges of X and Y highlighted. The correct output of the protocol is the path P leading to the leaf L. On the bottom is a possible 12-round execution of the protocol. Each step corresponds to one symbol over a linearly large alphabet. The highlighted steps correspond to correct edges (that move the protocol forward) appearing. The times $t(i)$ when the first i steps of P have been stated for the first time are also shown.

4.2 Analysis

Fix any run of the above protocol. We shall argue that if the protocol did not identify the correct vertex $v(X \cup Y)$, then the number of errors in transmissions must have exceeded $2R \cdot (1/4 - \epsilon)$.

For $i \in \{1, \ldots, R\}$, let $m(i)$ be the largest number such that the first $m(i)$ symbols of $D(a'_1, \ldots, a'_i)$ are equal to $a_1, \ldots, a_{m(i)}$ and the first $m(i)$ symbols of $D(b'_1, \ldots, b'_i)$ are equal to $b_1, \ldots, b_{m(i)}$. Set $m(0) = 0$.

Let P denote the set of edges along the unique path from the root of the tree to a leaf in the edges $X \cup Y$. Let $t(i)$ be the smallest j for which $E(a_1, \ldots, a_j) \cup E(b_1, \ldots, b_j)$ contains the first i edges of P. Set $t(i) = R + 1$ if no such j exists. Observe that by the definition of E, it must be the case that for any $j \geq t(i)$, $E(a_1, \ldots, a_j) \cup E(b_1, \ldots, b_j)$ contains the first i edges of P. Set $t(0) = 0$.

For $i \leq j$ in $[R]$, define $N(i, j)$ to be the number of transmission errors in the $[i, j]$ interval of the protocol:

$$N(i,j) = |\{k | i \leq k \leq j, C(a_1, \ldots, a_k) \neq a'_k\}| + |\{k | i \leq k \leq j, C(b_1, \ldots, b_k) \neq b'_k\}|$$

If $i > j$, define $N(i, j) = 0$.

We first show that if $i - m(i)$ is large, then the distance property of the tree code ensures that there must be lots of errors in the interval $[m(i) + 1, i]$:

Lemma 4. *$N(m(i) + 1, i) \geq (1 - \epsilon)(i - m(i))/2$.*

Proof. The lemma is trivial if $m(i) = i$, so let us assume that $m(i) < i$, and without loss of generality, assume that $a_{m(i)+1} \neq D(a'_1, \ldots, a'_i)_{m(i)+1}$. Since the tree code has distance $(1 - \epsilon)$, the hamming distance between $\overline{C}(a_1, \ldots, a_i)$ and $\overline{C}(D(a'_1, \ldots, a'_i))$ is at least $(1 - \epsilon)(i - m(i))$, and these two agree on the first $m(i)$ symbols.

Thus the only way these two vertices could be confused in the tree code is if the number of errors in the next $(i - m(i))$ symbols is at least $(1 - \epsilon)(i - m(i))/2$. □

Next we argue that if the first i rounds do not contain the first k edges on the correct path, then it must be the case that there is an inconsistency before the first $k - 1$ edges were announced.

Lemma 5. *For $i \geq 0$, $k \geq 1$, if $i + 1 < t(k)$, then $m(i) < t(k - 1)$.*

Proof. Without loss of generality, assume that k'th edge on the path P is an element of X. Suppose $m(i) \geq t(k - 1)$ and $i + 1 < t(k)$. Then it must be the case that the first $k - 1$ edges of P are all in A_{i+1}, since these edges have already been announced within the first $m(i)$ transmissions of both parties, yet the k'th edge is not included in A_{i+1}. The first party will announce this edge in round $i + 1$, contradicting the fact that $i + 1 < t(k)$. □

We can now use these two lemmas to get another bound on the number of errors:

Lemma 6. *For $i \geq -1, k \geq 0$, if $i + 1 < t(k)$, then $N(1, i) \geq (i - k + 1)(1 - \epsilon)/2$.*

Proof. We prove the statement by induction on $i + k$. For the base case, when $i \leq 0$ or $k = 0$, the bound is trivially true since the right hand side must be non-positive. For the general case, we have that

$$N(1, i) = N(1, m(i)) + N(m(i) + 1, i)$$

Note that this is true even when $m(i) = 0$ or $m(i) = i$. Lemma 4 implies that

$$N(m(i) + 1, i) \geq (i - m(i))(1 - \epsilon)/2.$$

By Lemma 5, we have that $m(i) < t(k - 1)$, so we can use the inductive hypothesis to bound

$$N(1, m(i) - 1) \geq (m(i) - 1 - (k - 1) + 1)(1 - \epsilon)/2 = (m(i) - k + 1)(1 - \epsilon)/2.$$

Summing these two bounds proves the lemma. □

Observe that if $m(R) \geq t(T)$, then at the conclusion of the protocol both parties have agreed on the correct value for $v(X \cup Y)$, since it must be the case that the only path connecting the root to a leaf must be the correct one, in both A_R and B_R. Thus the only way the protocol can fail is if $m(R) < t(T)$.

In this case, we have that $N(1, R) \geq N(1, m(R)) + N(m(R) + 1, R)$, which by Lemma 6 and Lemma 4 is at least $(m(R) - T)(1 - \epsilon)/2 + (R - m(R))(1 - \epsilon)/2 = (R - T)(1 - \epsilon)/2$. Thus the error rate is $N(1, R)/2R \geq (1 - \epsilon)^2/4 \geq 1/4 - \epsilon/2$.

5 Recovering from errors using a constant size alphabet

In this section, we prove Theorem 1.

Our protocol that uses a constant size alphabet is conceptually quite similar to the protocol from Section 4. As before, each prefix of transmissions of each of the parties shall encode a sequence of edges from the tree. We shall set $R = O(\lceil T/\epsilon \rceil)$ (the number of rounds in our protocol) to be large enough to make the analysis succeed. Define the alphabet

$$\Gamma = \{<, 0, 1, >, \emptyset\}.$$

Set $d = |\Gamma|$ and let $C : \Gamma^{\leq R} \to \Sigma^{\leq R}$ and $D : \Sigma^{\leq R} \to \Gamma^{\leq R}$ be the encoding and decoding functions of a d-ary tree code of distance $1 - \epsilon$.

Recall that $\Pi = \{1, 2, \ldots, R\} \times \{0, 1\}^{\leq 2}$ was the alphabet used in the previous section. In Section 4 every string $(z_1, \ldots, z_k) \in \Pi^k$ could potentially represent k edges. Here, we shall think of encoding these edges in the alphabet Γ as $< z_1 >< z_2 > \cdots < z_k >$. Namely, we replace each z_i with its binary representation, delimited with $<$ and $>$.

Fix a representation of Π using binary strings, with the property that the length of the encoding for (ℓ, s) is at most $c \log \ell$ for some constant c. To formally define our protocol, we need to show how to decode any string $a \in \Gamma^k$ to a set of edges from the tree. Fix such an a. For $i = 1, 2, \ldots, k$, we define $z_i \in \Pi$ as follows. If there exists a substring $a_j, a_{j+1}, \ldots, a_i$ such that $a_j =$ " $<$ ", $a_i =$ " $>$ " and these two symbols enclose a binary string, let (ℓ, s) denote the element of Π encoded by this binary string, and set $z_i = (j - \ell, s)$. If there is no such substring ending at a_i, set $z_i = (i, 00)$ so that it cannot encode any edge. Let E be the function mapping elements of Π^k to edges in the protocol tree, defined in Section 4, and let a encode the set of edges $E(z_1, \ldots, z_k)$.

Given such a string $a \in \Gamma^k$, we abuse notation and write $E(a)$ to denote the set of edges obtained in the process above. Observe that as before, $E(a)$ contains at most 1 edge out of every parent in the tree $\mathcal{T}$ and $E(a_1, \ldots, a_i) \subseteq E(a_1, \ldots, a_{i+1})$.

Our protocol for the pointer jumping problem is very similar to the protocol from the last section. The main difference is that an entire edge cannot be transmitted in a single transmission. So instead, the parties continue to transmit an edge if they were already transmitting the correct edge, and restart if they decide to transmit a different edge.

As before, the protocol has R rounds. In the i'th round, the first party (resp. second party) will compute a symbol a_i (resp. b_i) and transmit an element of Σ which is then received as a'_i (resp. b'_i). Exactly as in the last section, for every round i, define

$$A_i = \begin{cases} \emptyset & \text{if } i = 1, \\ E(a_1, \ldots, a_{i-1}) \cup (E(D(b'_1, \ldots, b'_{i-1})) \cap \mathcal{Y}) & \text{else.} \end{cases}$$

and

$$B_i = \begin{cases} \emptyset & \text{if } i = 1, \\ E(b_1, \ldots, b_{i-1}) \cup (E(D(a'_1, \ldots, a'_{i-1})) \cap \mathcal{X}) & \text{else.} \end{cases}$$

5.1 Protocol

1. For $i = 1, \ldots, R$,

   First Party If $v(A_i) \neq v(A_i \cup X)$, let e denote the edge out of $v(A_i)$ from X. Let a_i be the next symbol that needs to be transmitted to extend $a_1, \ldots, a_{i-1}$ to encode the edge e — namely if e is already being transmitted in $a_1, \ldots, a_{i-1}$, set a_i to be the next symbol of the transmission, else set $a_i =$ " $<$ ". If $v(A_i) = v(A_i \cup X)$, set $a_i = (R, 00)$, so that a_i cannot represent any edge. Transmit $C(a_1, \ldots, a_i)$.

   Second Party If $v(B_i) \neq v(B_i \cup Y)$ let e denote the edge out of $v(B_i)$ from Y. Let b_i be the next symbol that needs to be transmitted to extend $b_1, \ldots, b_{i-1}$ to encode the edge e — namely if e is already being transmitted in $b_1, \ldots, b_{i-1}$, set b_i to be the next symbol of the transmission, else set $b_i =$ " $<$ ". If $v(B_i) = v(B_i \cup Y)$, set $b_i = (R, 00)$, so that b_i cannot represent any edge. Transmit $C(b_1, \ldots, b_i)$.

2. The outcome of the protocol from the first (resp. second) party's point of view is the vertex $v(A_{R+1})$ (resp. $v(B_{R+1})$).

5.2 Analysis

The analysis is very similar to the analysis we used before. The main difference is that the transmission of an edge can involve a logarithmic number of rounds of communication. However, we shall argue that this does not hurt the communication of the protocol globally, because on average, the number of bits required to encode each offset is still constant. Fix any run of the above protocol. We shall argue that if the protocol did not identify the correct leaf of the tree, then the number of errors in transmissions must have exceeded $R(1/4-\epsilon/2)$. We define $m(i), t(i)$ exactly as in the last section. Recall that P denotes the set of edges along the unique path from the root of the tree to a leaf in the edges $X \cup Y$ and that $N(i,j)$ denotes the number of errors in transmission in rounds i through j.

Recall that $m(0)=t(0)=0$. It is easy to check that Lemma 4 holds for the new protocol, with exactly the same proof. Next we prove the analogue to Lemma 5.

Lemma 7. *For $i \geq 0, k \geq 1$, if $i+1 < t(k)$ then either $t(k-1) > i - c\log(i-t(k-1))$, or there exists j such that $m(j) < t(k-1)$ and $i - c\log(i-t(k-1)) < j \leq i$.*

Proof. Without loss of generality, assume that the k'th edge on the path P is an element of X. Suppose $t(k-1) \leq i - c\log(i-t(k-1))$ and $m(j) \geq t(k-1)$ for every round j satisfying $i - c\log(i-t(k-1)) < j \leq i$. Then it must be the case that the first $k-1$ edges of P are all in A_{j+1}, for every such j. Indeed, these edges have already been announced within the first $m(j)$ transmissions of both parties. The first party will then announce this edge, which will take at most $c\log(i-t(k-1))$ rounds of communication. Thus the k'th edge will be announced by round $i+1$, contradicting the assumption that $i+1 < t(k)$. □

We shall prove:

Lemma 8. *For $i \geq -1, k \geq 1$, $i+1 < t(k)$, then there exist numbers $\ell_1, \ell_2, \ldots, \ell_k \geq 0$ such that $\sum_{s=1}^{k} \ell_s \leq i+1$ and $N(1,i) \geq \frac{(i-k+1-\sum_{s=1}^{k} c\log(\ell_s+2))(1-\epsilon)}{2}$.*

Proof. We prove the lemma by induction on $i+k$. When $i \leq 0$ or $k=1$, the lemma is trivially true with $\ell_s = 0$ for all s. For the general case, by Lemma 7, there are two possibilities. The first is that $t(k-1) > i - c\log(i-t(k-1))$. In this case we set $\ell_k = i - t(k-1)$ and use the inductive hypothesis with $i = t(k-1)-1$ to show that there exist $\ell_1, \ldots, \ell_{k-1}$ with $\sum_{s=1}^{k} \ell_s \leq i+1$ such that

$$\begin{aligned} N(1,i) &\geq N(1, t(k-1)-1) \\ &\geq \frac{\left(t(k-1)-1-(k-1)+1-\sum_{s=1}^{k-1} c\log(\ell_s+2)\right)(1-\epsilon)}{2} \\ &= \frac{\left(i-(i-t(k-1))-k+1-\sum_{s=1}^{k-1} c\log(\ell_s+2)\right)(1-\epsilon)}{2} \\ &\geq \frac{\left(i-k+1-\sum_{s=1}^{k} c\log(\ell_s+2)\right)(1-\epsilon)}{2} \end{aligned}$$

The second possibility is that there exists j, such that $m(j) < t(k-1)$ and $i - c\log(i-t(k-1)) < j \leq i$. Set $\ell_k = i-j$. We have that $N(1,i) \geq N(1,m(j)-1) + N(m(j)+1, j)$. By the inductive hypothesis, there exist $\ell_1, \ldots, \ell_{k-1}$ so that $\sum_{s=1}^{k} \ell_s \leq m(j)+i-j \leq i+1$ and

$$\begin{aligned} N(1,m(j)-1) &\geq \frac{\left(m(j)-1-(k-1)+1-\sum_{s=1}^{k-1} c\log(\ell_s+2)\right)(1-\epsilon)}{2} \\ &\geq \frac{\left(m(j)-k+1-\sum_{s=1}^{k-1} c\log(\ell_s+2)\right)(1-\epsilon)}{2} \end{aligned}$$

By Lemma 4, $N(m(j)+1,j) \geq (j-m(j))(1-\epsilon)/2$. Thus

$$N(1,i) \geq \frac{\left(j-k+1-\sum_{s=1}^{k-1} c\log(\ell_s+2)\right)(1-\epsilon)}{2} \geq \frac{\left(i-k+1-\sum_{s=1}^{k} c\log(\ell_s+2)\right)(1-\epsilon)}{2}.$$

□

Finally we argue that Lemma 8 implies that $m(R) \geq t(T)$ unless the number of errors is larger than $2R(1/4 - \epsilon/2)$. Suppose $m(R) < t(T)$. We have that $N(1, R) \geq N(1, m(R) - 1) + N(m(R) + 1, R)$. If $m(R) < t(T)$, using the concavity of log, Lemma 8 implies that

$$\begin{aligned}\frac{N(1, m(R) - 1)}{2R} &\geq \frac{\Big(m(R) - T - \sum_{s=1}^{T} c \log(\ell_s + 2)\Big)(1 - \epsilon)}{2 \cdot 2R} \\ &\geq \frac{\Big(m(R) - cT \cdot \log\Big(\frac{\sum_{s=1}^{T} \ell_s + 2T}{T}\Big)\Big)(1 - \epsilon)}{4R} \\ &\geq \frac{(m(R) - cT \cdot \log(3R/T))(1 - \epsilon)}{4R}.\end{aligned}$$

By Lemma 4, $N(m(R) + 1, R) \geq (R - m(R))(1 - \epsilon)/2$. Thus $N(1, R)/2R \geq \frac{(R - cT \cdot \log(3R/T))(1-\epsilon)}{4R}$. Setting $R = O(T \cdot (c/\epsilon)\log(c/\epsilon))$ to be large enough, we get that $N(1, R)/2R \geq (1 - \epsilon)^2/4 \geq 1/4 - \epsilon/2$.

6 Only worst case inconsistent protocols can tolerate 1/4 error

In this section we show that no protocol can recover from an error rate of 1/4, as long as it has a natural property that we call *robustness*.

Note that for any protocol to recover form an error rate of ρ, it must be that as long as the total number of transmission errors is less than a ρ fraction, both parties have a consensus about whose turn it is to transmit based on their inputs and prior communication. A stronger requirement is to guarantee that both parties know whose turn it is to speak (and whether the protocol has terminated) for *any* number of errors in the transmissions. We call protocols that satisfy this requirement *robust*. Observe that all of our protocols for solving the pointer jumping problem were in fact robust, since the two parties send exactly one symbol from Σ in each step for a fixed pre-determined number of steps. We note that:

Claim 9. *If a protocol π is robust, then every execution of π must terminate in the same amount of time, and which player speaks at step i is determined by i and is independent of the inputs.*

Proof. Let π be any robust protocol. Suppose for the sake of contradiction that there are two vertices v_1, v_2 at the same depth k in the protocol tree which are inconsistent in terms of whether or not the protocol has terminated, or who the next speaker is. Let x, y' be inputs such that $\pi(x, y')$ passes through v_1 and x', y be inputs such that $\pi(x', y)$ passes through v_2. Now imagine what happens when the protocol is run with inputs x, y with transmission errors such that at every step, the symbols that Alice receives from Bob are consistent with $\pi(x, y')$, and the symbols Bob receives from Alice are consistent with $\pi(x', y)$. Then we see that in the k'th step the parties will be in inconsistent states, contradicting the fact that this was a robust protocol. □

Consider the following communication problem: Alice is given an input bit x, Bob is given an input bit y and the goal is for both parties to compute the parity $x \oplus y$. We show that no robust deterministic protocol for computing the parity can recover from 1/4 errors.

Claim 10. *No robust protocol of communication complexity T can compute $x \oplus y$ correctly and tolerate $\lceil T/4 \rceil$ errors.*

Proof. By Claim 9, every leaf in π has the same depth T. Further, the speaker at step i depends only on i. Assume without loss of generality that Alice speaks in $n \leq T/2$ of the rounds. Denote the indices of the steps where A speaks by $1 \leq i_1 < i_2 < \ldots < i_n \leq L$.

Now consider the following execution of π, from Bob's viewpoint. In every step that Bob speaks, Bob transmits his message consistent with his input being 0. In the first $\lfloor n/2 \rfloor$ times that Alice speaks, her transmission is received to be consistent with Alice having the input 0. In the remaining $\lceil n/2 \rceil$ transmissions of Alice, her transmission is received to be consistent with her having the input 1. From the point of view of Bob, this execution is consistent with an execution of π with inputs $x = 0, y = 0$ and $\lceil n/2 \rceil$ errors, and is also consistent with an execution of the protocol with $x = 1, y = 0$ and $\lfloor n/2 \rfloor$ errors. Thus Bob cannot compute $x \oplus y$ correctly by the end of the protocol if the number of errors is allowed to be as large as $\lceil n/2 \rceil \leq \lceil T/4 \rceil$. □

7 Encoding using the binary alphabet

It is possible to translate our results to the binary alphabet setting. To do this, we need to use a slightly different definition of tree codes. For a constant p, we label the edges of the tree with binary strings from $\{0,1\}^p$. Thus, the corresponding encoding function $C : [d]^{\leq n} \rightarrow \{0,1\}^p$ maps each vertex in the tree to a constant-length binary string that is included in the encoding of the path. As before, the distance of the tree code is α if for every u, v at the same depth, $\overline{C}(u)$ and $\overline{C}(v)$ differ in at least $\alpha \cdot \ell(u, v)$ coordinates.

We call such a tree code a tree code with binary alphabet. An easy analogue of Theorem 3 is the following:

Theorem 11. *For every $0 < \alpha < 1/2$ there exists a $p = O_{\alpha,d}(1)$ and a d-ary tree code of depth n that uses the binary alphabet and maps every symbol in $[d]^{\leq n}$ to a binary string of length p.*

Using Theorem 11, Theorem 2 follows in the same manner as Theorem 1 follows from the existence of ordinary tree codes. We omit the proof here.

Remark 12. *More generally, for each constant l and for each ϵ we can solve the pointer jumping problem with communication $O(T)$ even in the presence of a $(1/4) \cdot (1 - 1/l) - \epsilon$ fraction of errors, using the alphabet $[l]$.*

8 Conclusions and open problems

Our paper is an improvement on Schulman's [Sch96] work in that it handles a larger (and potentially the maximum) fraction of errors. In addition, our scheme has no large hidden constants, which would make it, in principle, practical if explicit efficient constructions of tree codes existed. To date, no constant-rate efficient constructions are known. There is a simple explicit construction [Sch03], but it only achieves a rate of $\Theta(1/\log n)$ for trees of depth n. Thus a major open problem is:

Open Problem 1: Construct an explicit constant-rate tree code with computationally efficient encoding and decoding.

Open Problem 2: Although we managed to show that robust protocols cannot tolerate a 1/4 error rate, we did not succeed in proving that arbitrary protocols must fail with this kind of error. What is the maximum error rate that can be tolerated?

Open Problem 3: Is there a binary error-correcting scheme that recovers from a β-fraction of errors with $\beta \geq 1/8$?

Finally, an interesting question to ask is whether any kind of recovery is possible if we lower our requirement to correctly compute the correct outcome of the original protocol. In the one-way communication setting a rich body of work on list decoding with up to twice the error exists (see, e.g. [Sud00] for a survey). It is extremely interesting to see how list-decoding concepts would apply to interactive error-correction, and in particular to identify the right notion of list-decodable tree codes that would produce such results. This direction may also hold the key to Open Problems 2 and 3 above.

9 Acknowledgments

We would like to thank Trinh Huynh, Madhu Sudan and Avi Wigderson for their useful comments.

References

[KN97] Eyal Kushilevitz and Noam Nisan. *Communication complexity.* Cambridge University Press, Cambridge, 1997.

[PWJ72] W. W. Peterson, E. J. Weldon, and Jr. *Error-Correcting Codes.* MIT Press, Cambridge, Massachusetts, 1972.

[Sch96] Leonard J. Schulman. Coding for interactive communication. *IEEE Transactions on Information Theory*, 42(6):1745–1756, 1996.

[Sch03] L. Schulman. A postcript to "Coding for Interactive Communication", 2003. http://www.cs.caltech.edu/ schulman/Papers/intercodingpostscript.txt.

[Sha48] Claude E. Shannon. A mathematical theory of communication. *Bell System Technical Journal*, 27, 1948. Monograph B-1598.

[Sud00] M. Sudan. List decoding: Algorithms and applications. *Theoretical Computer Science: Exploring New Frontiers of Theoretical Informatics*, pages 25–41, 2000.