Electronic Colloquium on Computational Complexity, Report No. 93 (2017)

Interactive Coding Over the Noisy Broadcast Channel

Klim Efremenko[*] Gillat Kol[†] Raghuvansh R. Saxena[‡]

Abstract

A set of n players, each holding a private input bit, communicate over a noisy broadcast channel. Their mutual goal is for all players to learn all inputs. At each round one of the players broadcasts a bit to all the other players, and the bit received by each player is flipped with a fixed constant probability (independently for each recipient). How many rounds are needed?

This problem was first suggested by El Gamal in 1984. In 1988, Gallager gave an elegant noise-resistant protocol requiring only $\mathcal{O}(n \log \log n)$ rounds. The problem got resolved in 2005 by a seminal paper of Goyal, Kindler, and Saks, proving that Gallager's protocol is essentially optimal.

We revisit the above noisy broadcast problem and show that $\mathcal{O}(n)$ rounds suffice. This is possible due to a relaxation of the model assumed by the previous works. We no longer demand that exactly one player broadcasts in every round, but rather allow any number of players to broadcast. However, if it is not the case that exactly one player chooses to broadcast, each of the other players gets an adversely chosen bit.

We generalized the above result and initiate the study of interactive coding over the noisy broadcast channel. We show that any interactive protocol that works over the noiseless broadcast channel can be simulated over our restrictive noisy broadcast model with constant blowup of the communication.

[*]Tel-Aviv University
[†]Princeton University
[‡]Princeton University

ISSN 1433-8092

1 Introduction

In this paper, we initiate the study of interactive coding over the wireless noisy broadcast channel. In the interactive coding problem, one is given an interactive communication protocol between several parties communicating over a (noiseless) channel. The goal is to design a noise-resilient protocol that "simulates" the original protocol (e.g., computes the same function) over a noisy channel, with as little communication overhead as possible. The interactive coding problem attracted a lot of attention in recent years, and was studied for varies two-party settings, as well as for the multi-party case where players communicate over a *point-to-point* network (private channels). We study the interactive coding problem over a new, restrictive, noisy broadcast model. We show that even in this model, every protocol can be simulated with a constant multiplicative overhead.

The *noisy broadcast model* was proposed by El Gamal [Gam87] in 1984, as a simple model allowing the study of the effect of noise in a highly distributed system. This model considers n parties, each holding a private input bit x_i, and the goal of the parties is to compute a function $f(x_1, \cdots, x_n)$ of all n inputs, using as few communication rounds as possible. Communication is carried out in a sequence of synchronous rounds. In every round, one of the parties broadcasts a bit over the channel. Each of the other parties independently receives the broadcast bit, with probability $1 - \epsilon$, or the complement of the bit, with probability ϵ, where ϵ is some fixed noise parameter. At the end of the protocol, all players need to know $f(x_1, \cdots, x_n)$.

In 1988, Gallager [Gal88] gave such a noise-resistant protocol for the identity function (all players need to learn all input bits), that requires only $\mathcal{O}(n \log \log n)$ broadcast rounds and errs with some small sub-constant probability. Gallager's result was shown to be tight in the beautiful 2005 paper by Goyal, Kindler and Saks [GKS08].

1.1 Our Results

We revisit the above mentioned works in the more general framework of interactive coding, that not only considers n-bit functions, but attempts to simulate the execution of any interactive communication protocol with any input size. Simulating a general n-round interactive communication protocol with n-players over a noisy channel is more challenging than simulating the (naive) protocol for identity on n-bits over the a noisy channel. The reason is that every message broadcast by the communication protocol may depend on all previous messages. There are no such dependencies between the messages sent by the identity protocol, as it merely communicates the n input bits one-by-one.

Towards the goal of round-efficient interactive coding, we observe that the impossibility result of [GKS08] crucially depends on the assumption that the model is *non-adaptive*, that is, the player broadcasting in each of the rounds is known in advance and is only a function of the round number (the player cannot depend on the inputs or the noise in the channel). Non-adaptivity is assumed because it prevents multiple players from broadcasting in the same round.

To bypass the $\Omega(n \log\log n)$ lower bound of [GKS08], we relax the demand that the order in which the players broadcast is known in advance. We define an *adaptive* noisy broadcast model, we call the (n, ϵ)-noisy broadcast model, where any number of players may broadcast at the same round. If exactly one player broadcasts in a given round, then, as before, the players each get an independent ϵ-noisy copy of the broadcast bit. We emphasize that the identity of the broadcasting player is not revealed to the other players. However, if in a given round it is not the case that exactly one player broadcasts, all players receive adversarially chosen bits. That is, in case of collision (two or more parties broadcast in the same round) and in the case of silence (no player is broadcasting), an adversary chooses the bits received by the players, where each player may get a different bit. We assume that the adversary knows all the information about the execution of the protocol, including the players' inputs and randomness, and all their received transcripts. Our model tries to mimic the phenomenon that the "superposition" of two signals cannot be used to conclude anything about any one of the two signals. Furthermore, it rules out communication by silence ("signaling"). We require a simulation protocol to work against any strategy of the adversary.

Various adaptive (noiseless) broadcast models are extensively studied in the context of wireless radio distributed communication, as these describe real-life scenarios (e.g., radio transmitters, mobile distribution towers, etc.). However, these models usually either assume that collision is detectable, or that collision is received by the players as silence, or that in the case of collision, exactly one of the messages broadcast is received (see [Pel07] for a survey). Our model is more restrictive than all of the above, as both collision and silence are fully adversarial. In particular, any protocol that works in our model will also work in the noisy versions of all of the above models.

Our main result is that even in our restrictive (n, ϵ)-noisy broadcast model, any protocol can be simulated with a constant blow-up in the communication.

Theorem 1.1. *Let $\varepsilon \in (0, \frac{1}{10})$. For any non-adaptive randomized n-party protocol Π that takes T rounds assuming an n-party noiseless broadcast channel, there exists a randomized adaptive coding scheme that simulates π over the (n, ϵ)-noisy broadcast channel, that takes $\mathcal{O}(T)$ rounds, and errs with sub-constant probability in T.*

Designing general simulation protocols over the (n, ϵ)-noisy broadcast channel poses new challenges. To overcome these, we combine tools from interactive coding (e.g., tree codes), wireless distributed computing (e.g., version of the celebrated Decay protocol [BGI92]), and also develop new techniques.

1.2 Related Works

As mentioned above, Gallager was the first to design non-trivial noisy broadcast protocols. He considered the identity and the parity functions and gave $\mathcal{O}(n \log\log n)$ protocols for both [Gal88]. Noise-resilient protocols for varies other n-bit functions where studied by [KM98, FK00, New04, GKS08], showing that, for some interesting functions, $\mathcal{O}(n)$ rounds protocols are possible, even in the model where the noise rate is not fixed.

The field of interactive coding was introduced in a seminal paper of Schulman [Sch92]. Various aspects of two-party interactive coding (such as computational efficiency, interactive channel capacity, noise tolerance, list decoding, different channel types, etc.) were considered in recent years [Sch93, Sch96, GMS11, BR11, BKN14, Bra12, KR13, MS14, Hae14, BE14, GMS14, GH15, GHK+16, EGH15, BGMO16], to cite a few. In certain two-party settings, it is known that adaptive models may allow for better interactive coding schemes [Hae14, GHS14] (note, however, that the improvement in these cases is only by a constant).

Multi-party interactive coding over a point-to-point network was first studied by [RS94], and more recently by [ABE+16, BEGH16]. We mention that the noisy-broadcast setting is very different from the point-to-point case, as in the broadcast setting a player can only send the *same* bit to *all* other players.

2 Proof Sketch

2.1 High Level Overview

The main property of the adaptive model that we are exploiting in our simulation protocol, is that if a player detects an error, he can let the rest of players know *immediately*, and the corresponding bits can be retransmitted. In our protocol, we designate one of the players to be the leader, and essentially all communication will be routed through this leader. This is done by having the leader repeat every message that he receives in an "error-robust" way. More formally, the leader encodes every received bit using a tree code (see subsection 3.4), and broadcasts the encoded message. The tree code property ensures that if the leader receives an "incorrect" bit from some player, this player will detect the error within a constant number of rounds (in expectation) and will be able to signal this to the leader. We note that the leader may receive an incorrect bit, either due to an error in the transmission of this bit from the player to the leader, or since the player received previous bits incorrectly, thus computed an incorrect value.

We show that this mechanism can be implemented such that the probability that after t rounds, a player still did not realize that the bit the leader received from him was incorrect, is exponentially small in t. Therefore, we do not need to check with all the players whether the leader received their bit correctly, but it suffices to check this with the few last players to broadcast. More precisely, our protocol runs a consistency check every 2^j rounds (for every $j \in \mathbb{N}$). This check involves the last 2^j players to broadcast. We will show that the consistency check requires $\text{poly}(j)$ rounds, therefore it decreases the rate of our protocol by only a constant.

2.2 From Non-Adaptive Protocols to Correlated Pointer Jumping

We introduce the n-player correlated pointer jumping (CPJ) problem, and claim that it is "complete" for interactive coding over the noisy broadcast channel. A similar claim is known to hold for the two party case [BR11].

Consider a non-adaptive communication protocol Π between n players connected via a noiseless broadcast network. Since Π is non-adaptive, it specifies the player p_i that broadcasts in round i. Along with p_i, Π also specifies a function $g_i : \{0,1\}^{i-1} \times \mathcal{X} \to \{0,1\}$ that takes a transcript π_{i-1} of length $i-1$ and a private input $x \in \mathcal{X}$ to a single output bit. At round i, player p_i broadcasts the output bit $b_i = g_i(\pi_{i-1}, x)$ over the network. To simplify notation, we used the same input set $\mathcal{X}$ for all the n players. Since the channel is noise-free, all the n players receive a copy of b_i and use it extend their transcript $\pi_i = \pi_{i-1} \| b_i$. Player p_{i+1} can then broadcast and the protocol goes on.

Consider now an instance of the n-player CPJ problem. This instance is defined by a complete binary tree of depth d. The edges of this tree are labelled by elements in $\{0,1\}$. Every layer in this tree belongs to one of n players. We use q_i to denote the player that owns the nodes in layer $i \in [d]$. Every player gets as an input one bit b_v for every node v he owns. Since there are 2^{i-1} nodes in layer i, the bits b_v for v in layer i can be specified by a function $f_i : \{0,1\}^{i-1} \to \{0,1\}$ known only to player q_i. The bit b_v determines one of the two children of the node v. We call this child the 'correct' child. The root of the tree is assumed to be correct by default. The goal of the n players is to compute the (unique) path that has only correct nodes. This will be referred to as the correct path.

We claim that a communication protocol Π is equivalent to a CPJ instance of depth T, where T is the number of bits communicated in an execution of Π. The equivalence maps player p_i broadcasting in round i in Π to the player q_i who owns all the nodes at depth i in the CPJ tree. The functions f_i for the CPJ instance are created so as to satisfy $f_i(\pi_{i-1}) = g_i(\pi_{i-1}, x), \forall \pi_{i-1}$, where x is the input for Π. It can be proved that the correct path will be the transcript of the protocol Π.

We restrict our attention to CPJ instances of depth n where each layer belongs to a different player. We number the players from 1 to n so that layer i belongs to player i for all $i \in [n]$. We describe a protocol that finds the correct path for the CPJ instance over the (n, ϵ)-noisy broadcast model. Our protocol would require a number of broadcasts that is linear in n, the depth of the CPJ instance. Since CPJ instances are equivalent to noise-free broadcast protocols, this is actually a constant rate interactive coding result. Our assumption about the depth n of the CPJ tree being equal to the number of players is for simplicity. Indeed, the same ideas would extend to arbitrary depth trees where each layer is owned by an arbitrary player $i \in [n]$.

2.3 A Protocol for CPJ Over the Noisy Broadcast Model

We build our noise-tolerant CPJ protocol based on a protocol in the noise-free environment. In the absence of noise, the simplest protocol would start from the root v_1 of the tree. Since the root (layer 1) belongs to player 1, player 1 will broadcast b_{v_1}. This bit would be heard correctly by all the players. b_{v_1} identifies a correct node v_2 in layer 2. Player 2 then broadcasts b_{v_2} which is, again, heard correctly by all the players. The protocol proceeds until a length n path has been identified. Since the players only broadcast correct bits, the path is, by definition, correct.

As is evident, this protocol relies on the correct reception of each and every bit by all the n players. This can be achieved by broadcasting just once in the noise-free setting. In the noisy environment however, ensuring correct reception with a constant probability by all the players requires $\Omega(\log n)$ broadcasts. The naive interactive coding protocol, thus, incurs an overhead of $\Omega(\log n)$.

To avoid this $\Omega(\log n)$ slowdown, we use the observation that not all players need to know every bit *immediately* after it was broadcast. Consider the first bit that is broadcast. For large i, since player i only broadcasts in round i, he wouldn't need to know the first bit until $i-1$ addition rounds have passed. Thus, trying to convince player i of the value of the first bit immediately after its broadcast is not necessary or optimal.

We develop this idea further. Consider a player i such that $i \in [2^j, 2^{j+1})$. Player i will definitely not broadcast in the first $2^j - 1$ rounds and hence, they do not *need* to know anything until then. We design our protocol in a way such that for all j, the first 2^j rounds only convince the first 2^{j+1} players. As the protocol proceeds, j starts taking higher and higher values and eventually, we will be able to convince every player about every bit.

More formally, we define a sequence of protocols $\{\mathcal{A}_j\}_{j\geq 0}$. The protocol $\mathcal{A}_0$ only involves player 1 and the leader. In this protocol, player 1 broadcasts b_{v_1} and then they check if the leader received b_{v_1} correctly. For $j > 0$, the protocol $\mathcal{A}_j$ first runs $\mathcal{A}_{j-1}$. Let s_1 be the path that was successfully transmitted by this execution $\mathcal{A}_{j-1}$. We ensure that $|s_1| = \Omega(2^{j-1})$ and the probability that a given player $i \in [2^j]$ doesn't have the correct value of s_1 is $2^{-\Omega(j)}$. After this, the players run $\mathcal{A}_{j-1}$ again. This second invocation starts at the point where the previous invocation ended, i.e., after the first $|s_1|$ edges. The output s_2 of this protocol will again be correct for players in $[2^j]$ with probability at least $1 - 2^{-\Omega(j)}$.

After the two executions of $\mathcal{A}_{j-1}$, the protocol $\mathcal{A}_j$ runs a check phase. This check phase is longer than the check phase in $\mathcal{A}_{j-1}$ and is designed to convince 2^{j+1} players. The check phase decides which prefix of the path $s_1 \| s_2$ was communicated correctly. The expected length of this prefix is roughly double the expected length of s_1 (and also s_2). This length is the 'progress measure' of our protocol. Since it doubles each time j increases by 1, it stays within a constant factor of the number of bits transmitted. The induction continues and j grows to convince all the players about all the bits.

2.4 The Check Phase

The protocol $\mathcal{A}_j$ is designed to convince twice as many players as the protocol $\mathcal{A}_{j-1}$. Also, the probability that $\mathcal{A}_j$ fails is half the probability that $\mathcal{A}_{j-1}$ fails. The protocol $\mathcal{A}_j$ gives these improved guarantees because it uses longer check steps. A check step in $\mathcal{A}_j$ involves 2^j players, each of which has an input string $v_i, i \in [2^j]$, corresponding to his belief of the position in the CPJ tree that the simulation protocol has reached.

In a check round, the players compare their bit strings with each other. In order to do this, one of the players (the leader) broadcasts a hash of length $\mathcal{O}(j)$ of their string. This hash is repeated enough times ($\mathcal{O}(j)$ times) to ensure correct reception by all the 2^j players with probability at least $1-\mathcal{O}(2^{-j})$. All the players $i \in [2^j]$ then compare the hashes received from the leader with their hashes of the string v_i. If the hashes match, then the strings are the same with high probability. Otherwise, the strings are most probably different.

Player i computes the bit b_i that is 1 if and only if the hashes of v_i are the different from the hashes he received from the leader. If there exists a player i such that b_i is 1, then the input v_i of player i is different from the leader's string. Thus, by computing the OR of the bits b_i, the players can detect if their strings are consistent. We describe the protocol `OR` for computing the OR function in subsection 2.5.

Due to the noise present in the channel, the strings v_i will not be identical for all players i almost always. The check round will, thus, almost always fail and the players will not make progress. To avoid this, we compute the longest prefix that is common to all the strings v_i. We show that the length of this prefix will typically be a constant fraction of the length of v_i. Thus, in 2^j rounds, the players would agree on $\Omega(2^j)$ bits. The longest common prefix (LCP) is computed by a binary search on the length of the strings. Since the check procedure is efficient (polynomial in j broadcasts) and the prefix output is always a constant fraction of $|v_i|$, our protocol requires $\mathcal{O}(n)$ broadcasts to compute the length n output of CPJ.

The longest common prefix is an indicator of the part of the CPJ tree that was simulated correctly. After the check step, the players go back to the location in the CPJ tree indicated by the result of the LCP and start simulating from there on. Thus, all the players that may need to broadcast their bits in the near future, need to know the result of the LCP. For this reason, we ensure that the check rounds convince not only the (at most) 2^j players that broadcast their inputs in the two executions of $\mathcal{A}_{j-1}$, but all the first 2^{j+1} players.

Another issue is that each of the other players (player $i \in [n] \setminus [2^{j+1}]$) will eventually need to know the correct location in the CPJ tree to be able to broadcast his relevant input bit, thus will need to know the result of the LCP and all the input bits that were broadcast during the two executions of $\mathcal{A}_{j-1}$. We do not have the budget to allow all the players to participate in the check step, nor can we afford to repeat all the broadcast inputs many times. What we do instead is have the leader repeat all the information he receives (his perception of the input bits sent by the different players, as well as the LCP result) over a tree code (see subsection 3.4 and subsection 2.1). The usage of a tree code ensures that with time, each of the players will be able to retrieve the correct transcript of the protocol.

2.5 The `OR` protocol

We now describe our protocol for the OR function. In this setting, each player $i \in [2^j]$ has a private input bit b_i. The players work to compute $or = \vee_{i=1}^{2^j} b_i$. Since $or = 1$ if and only if there exists i such that $b_i = 1$, we partition the set of 2^j players into two classes based on b_i. If $b_i = 1$, player i is called a 1-player. Otherwise, he is called a 0-player. The 1-players and the 0-players play complementary roles in the `OR` protocol.

A 1-player i knows his input $b_i = 1$, and thus can compute $or = 1$. His job is to convince every other player about his presence. If one of the 1-players is able to tell all the other players that their bit is 1, the other players will be able to compute $or = 1$. On the other hand, the 0-players know that their bit will not affect the final outcome and try to listen to the 1-players broadcasting, if any.

Since we assume the broadcast channel to be adversarial when more than one player (or no player) is broadcasting, if the 1-players keep broadcasting together, the bits received by all the players would be adversarial. To avoid this, the 1-players follow a random-back-off strategy, as in the Decay protocol [BGI92]. The players broadcast independently with probability α. They start with $\alpha = 1$, and decrease α by $1/2$ at each step. When α is roughly, the inverse of the number of 1-players, then in expectation, only one 1-player will broadcast and convince all the 0-players. The expectation argument can be converted to a high probability one by repeating enough times.

The random-back-off strategy described above is implemented in the protocol `Advertising` (see Algorithm 1) that requires $\mathcal{O}(j^3)$ broadcasts. It ensures that if a 1-player exists, they are able to communicate their presence to all the other players with high probability. However, if all players are 0-players, all the players will stay silent throughout these rounds and all the bits received would be adversarial. To deal with this case, we require every player broadcasting during `Advertising` to broadcast their identity. The identity of player i is just the binary representation of i, and can be communicated using at most $\mathcal{O}(j)$ bits. During the execution of `Advertising`, the leader will collect $\mathcal{O}(j^2)$ id's.

The players will then run the protocol `Inquiry` (see Algorithm 2). In this protocol, the leader will ask each of the $\mathcal{O}(j^2)$ identities he collected for their private input bit[1]. The players will respond with their private input bit. If there exist 1-players, one of them, say player i, would have successfully broadcast the identity i during `Advertising`. When the leader queries i, he will respond with 1. If all the players were 0-players, all the responses to the leader's queries will be 0. The players can, thus, output the OR of all the responses to compute or.

It is crucial in this protocol that `Inquiry` is "adversary free". The leader's queries are adversary free because he broadcasts in pre-determined rounds. This means that all players receive a noisy version of the query. Correct reception except with probability at most $\mathcal{O}(2^{-j})$ can be ensured by using error correcting codes. If all players receive the query correctly, exactly one player (the player whose identity was queried) would respond making

[1]These identities might be adversarial. This doesn't matter as long as they are in $[2^j]$.

the response adversary free. This response is again encoded to ensure correct reception. Since all the queries and their responses are received correctly by all players in $[2^j]$, we are able to prove that the output of this protocol is correct.

3 Preliminaries

3.1 Notation

Throughout this paper, we use A^k to denote the k-fold cartesian product of a set A with itself, *i.e.* $A^k = \underbrace{A \times A \times \cdots \times A}_{k \text{ times}}$. We define $A^{\leq n} = \bigcup_{i=1}^{n} A^i$. The set of all natural numbers less than or equal to n is denoted by $[n]$.

We denote by ε the empty string (the string of length zero). For a string s and $i, j \in \mathbb{N}$, we denote by $|s|$ the length of s, and by $s[i : j]$ the substring from position i to position j (both inclusive). If $i > j$, then $s[i : j] = \varepsilon$. If $|s| < j$, then we first pad s with zeros to be of length i, and then take the slice. We sometimes abbreviate $s[i : j-1]$ as $s[i : j)$; $s[i+1 : j]$ as $s(i : j]$ and so on.

Let S be a set of bit strings. We denote by $l(S)$ the longest prefix that is common to all strings in the set S. That is, $l(S)$ is a prefix of all the strings in S, and no string of length greater than $|l(S)|$ is a prefix of all strings in S. If $S = \{s_1, \ldots, s_m\}$, we sometimes write $l(s_1, \ldots, s_m)$ to denote $l(S)$. Given a vertex v in a binary tree, we often view v as the bit-string that corresponds to the path from the root to v. Hence, for two vertices v and u, $l(v, u)$ is the lowest common ancestor of v and u. For two bit strings s and t, we denote by $s \| t$ the concatenation of s and t.

3.2 Probability

We use the following formulation of Chernoff bound.

Lemma 3.1 (Multiplicative Chernoff bound)**.** *Suppose $X_1, \cdots, X_n$ are independent random variables taking values in $\{0, 1\}$. Let X denote their sum and let $\mu = \mathbb{E}[X]$ denote the sum's expected value. Then,*

$$\begin{aligned}
\Pr[X \geq (1+\delta)\mu] &\leq e^{-\frac{\delta^2\mu}{3}}, \qquad \forall 0 < \delta < 1,\\
\Pr[X \geq (1+\delta)\mu] &\leq e^{-\frac{\delta\mu}{3}}, \qquad \forall 1 \leq \delta,\\
\Pr[X \leq (1-\delta)\mu] &\leq e^{-\frac{\delta^2\mu}{2}}, \qquad \forall 0 < \delta < 1.
\end{aligned}$$

The following simple lemma considers error reduction by repetition.

Lemma 3.2 (Majority lemma)**.** *Let $\epsilon < 1/2$, $f > 1$ and $b \in \{0, 1\}$. There exists a constant c depending on ϵ, such that the majority of cf independent ϵ-noisy copies of b is b, except with probability at most* $\exp(-f)$.

Proof. For $i \in [cf]$, let X_i be a random variable that is 0 if the i^{th} noisy copy of b is indeed b, and 1 otherwise. For $i \in [cf]$, we have $\Pr[X_i = 1] = \epsilon$. Let $X = \sum_{i \in [cf]} X_i$. The majority is incorrect only if $X \geq cf/2$. By the Chernoff bound (Lemma 3.1),

$$\Pr[X \geq cf/2] \leq \exp(-\tfrac{\delta^2 \epsilon cf}{3}),$$

where $\delta = \min\{\frac{1}{2\epsilon} - 1, 1\}$. Setting $c = \frac{3}{\delta^2 \epsilon}$ gives the result. □

3.3 Error Correcting Codes

We will use the existence of the following error correcting codes. The following version is derived from [GKS08].

Theorem 3.3. *For $\gamma \in (0, 1/2)$, there is an integer $K_1 = K_1(\gamma)$ such that for all positive integers t and each $K \geq K_1$, there exists a code $C_t \subset \{0,1\}^{Kt}$ of size 2^t, such that for all $v, w \in C_t$ with $v \neq w$, $d(v, w) \geq \gamma K t$, where $d(v, u)$ denotes the Hamming distance between v and w.*

Let $\gamma \in (0, 1/2)$. In what follows, we let $E_{t,\gamma} : \{0,1\}^t \to \{0,1\}^{K_\gamma t}$ stand for the encoding function of the error correcting code promised by the above theorem for γ, and let $D_{t,\gamma} : \{0,1\}^{K_\gamma t} \to \{0,1\}^t$ be the corresponding decoding function. Our protocols use the following lemma.

Lemma 3.4. *Let $\epsilon \in (0, 1/8)$, $\gamma = 4\epsilon$, and $x \in \{0,1\}^{\log n}$. Suppose that $\tilde{E}$ is a noisy copy of $E_{\log n,\gamma}(x)$ obtained by flipping each of the $K_\gamma t$ bits of $E_{\log n,\gamma}(x)$ independently with probability ϵ. Then, the probability that $D_{\log n,\gamma}(\tilde{E}) = x$ is at least $1 - n^{-30}$.*

Proof. Let $K = K_\gamma$, and assume without loss of generality that $K > \frac{1000}{\epsilon}$. It holds that $D_{\log n,\gamma}(\tilde{E}) = x$ unless $d(\tilde{E}, E_{\log n,\gamma}(x)) > 2\gamma K t$. For $i \in [K \log n]$, define a random variable X_i to be 1 if $\tilde{E}$ and $E_{\log n,\gamma}(x)$ differ on the i^{th} bit, and 0 otherwise. Let $X = \sum_{i=1}^{K \log n} X_i$. Since the noise in each coordinate is independent, by the Chernoff bound (Lemma 3.1),

$$\Pr[X \geq 2\epsilon K \log n] \leq \mathrm{e}^{-\frac{\epsilon K \log n}{3}} \leq n^{-30}.$$

□

3.4 Tree Codes

Tree codes were introduced by Schulman [Sch93]. These are powerful "online" error correcting codes that work in an environment where the input is streaming. Thus, can be used towards the goal of reliable communication over noisy channels. We use the following version defined in [BR11].

Definition 3.5 ((d, α, Σ)-tree code)**.** *A d-ary tree code of depth n and distance $\alpha > 0$ over alphabet Σ is defined using an encoding function $C : [d]^{\leq n} \to \Sigma$ that satisfies the following property:*

Define $\overline{C}(v_1, v_2, \cdots, v_k)$ as $C(v_1) \| C(v_1, v_2) \| \cdots \| C(v_1, v_2, \cdots, v_k)$. Let u and v be any two strings of length $k \leq n$. Let $k_0 = k - |l(u, v)|$. Then, $\overline{C}(u)$ and $\overline{C}(v)$ differ in at least αk_0 coordinates.

Note that the way $\overline{C}$ is defined implies that $\overline{C}(u)$ and $\overline{C}(v)$ are of length k and share a prefix of length at least $k - k_0$. Thus, the definition implies that at least a α fraction of the remaining coordinates are different in both strings. Tree codes with larger values of α are more powerful.

Encoders C that satisfy this property in Definition 3.5 are called tree codes because they can be viewed as a d-ary tree of depth n, where each edge is labelled with an element from Σ. In this setting, each element s in $[d]^{\leq n}$ can be seen as defining a path from the root to some node in the tree, and $C(s)$ is just the symbol on the last edge in this path. The tree code condition implies that for any two leaves u and v at the same depth and $w = l(u, v)$, at least α fraction of the symbols on the path from w to u differ from the corresponding symbols on the path from w to v. The following theorem was first proved by [Sch96]:

Theorem 3.6. *For every $n, d \in \mathbb{N}$, and $0 < \alpha < 1$, there exists a $(d, \alpha, [d^{\mathcal{O}_\alpha(1)}])$-tree code of depth n.*

We now introduce the decoder for tree codes that we will use. The decoding function $D : \Sigma^{\leq n} \to [d]^{\leq n}$ takes the closest string σ in $\Sigma^{\leq n}$ (in terms of Hamming distance) that lies in the range of $\overline{C}$ and returns $\overline{C}^{-1}(\sigma)$. If two strings differ in their length, then we consider their Hamming distance to be ∞. The following theorem formalizes the property that we want a tree code to satisfy.

Theorem 3.7. *Fix $\alpha > 2\epsilon > 0$. Let C and $\overline{C}$ define a (d, α, Σ)-tree code of depth n. Let $k \leq n$ and consider any string $s \in [d]^k$. Let $\tilde{C}(s)$ be $\overline{C}(s)$ where each symbol is replaced with a random symbol from Σ independently with probability ϵ. Let $t = D(\tilde{C}(s))$ where D is the decoding function. Then, for any $k_0 \in \mathbb{N}$,*

$$\Pr\big(|l(s, t)| \leq k - k_0\big) \leq K_1 \exp(-K_2 k_0),$$

for some constants K_1 and K_2 depending on ϵ and α.

Proof. Define the random variable $l = l(s, t)$ and consider the event $l = l_0$. Since $\overline{C}$ defines a tree code, $\overline{C}(s)$ and $\overline{C}(t)$ differ in at least $\alpha(k - l_0)$ places. Furthermore, all of these differences are in the last $k - l_0$ coordinates. A Hamming distance based decoder would decode $\tilde{C}(s)$ to t only if at least $\alpha(k - l_0)/2$ indices were affected by noise. Define indicator random variables $\{X_i\}_{i=1}^{k}$ such that X_i is 1 if and only if index i is corrupted by noise. Let

$X_{>j} = \sum_{i=j+1}^{k} X_i$. An application of Lemma 3.1 gives:

$$\begin{aligned}
\Pr[l = l_0] &\leq \Pr[X_{>l_0} \geq \alpha(k-l_0)/2] \\
&= \Pr\left[X_{>l_0} \geq (1+\delta)\mathbb{E}[X_{l_0}]\right] \\
&\leq \exp\left(\tfrac{-\delta^2 \mathbb{E}[X_{l_0}]}{3}\right) = \exp\left(\tfrac{-\delta^2 \epsilon (k-l_0)}{3}\right),
\end{aligned}$$

where $\delta = \min\left(\frac{\alpha}{2\epsilon} - 1, 1\right)$ is a constant. This directly gives,

$$\begin{aligned}
\Pr\left[l(s,t) \leq k - k_0\right] &= \sum_{i=k_0}^{k-1} \Pr\left[l(s,t) = k - i\right] \\
&\leq \sum_{i=k_0}^{k-1} \exp\left(\tfrac{-\delta^2 \epsilon i}{3}\right) \leq \sum_{i=k_0}^{\infty} \exp\left(\tfrac{-\delta^2 \epsilon i}{3}\right) \\
&= \frac{\exp\left(\frac{-\delta^2 \epsilon k_0}{3}\right)}{1 - \exp\left(-\frac{\delta^2 \epsilon}{3}\right)}.
\end{aligned}$$

□

4 Our Framework

Each subsection here discusses an important feature in our protocol. We refer to this section extensively throughout the paper.

4.1 The Leader

A major difficulty in designing adaptive protocols in our (n, ϵ)-noisy broadcast model is to co-ordinate the broadcasts by all the players. This is difficult because the noise injected in the transcript received by the players is independent of each other. Thus, two different players might have entirely different views of the protocol. However, it is crucial to ensure that most rounds have only one player broadcasting. If this is not the case in some round, the bits received by all the players in that round are adversarial.

This important task is designated to a special player, called the "leader" in our protocols. Almost all the communication in all our protocols will be routed through the leader. For the rest of the text, we assume that the leader is an extra player (player $n+1$), but the same arguments hold if the leader is one of the original players.

We will make sure that the leader is broadcasting in *pre-determined rounds*, that is, in rounds whose number is fixed in advance, and does not depend on the player's inputs and randomness and on the noise in the channel. Since all the other players in $[n]$ know these rounds (they are specified by the protocol), they would never broadcast in these rounds. We sometimes call such pre-determined rounds "adversary free". The name reflects the fact that the bit received by all the players in these rounds in noisy, but not adversarial (as only the leader was broadcasting).

Since the leader's broadcasts are adversary free and the players only take instructions from the leader, important parts of our protocol are adversary free. Almost throughout, the players only broadcast after the leader asks them to, to ensure synchronization. The leader will only ask one player at any given time, making that player the only person broadcasting.

4.2 Global Tree Code

We solve the CPJ problem via a sequence of protocols $\{\mathcal{A}_j\}_{j\geq 0}$ where each protocol $\mathcal{A}_j$ invokes the protocol $\mathcal{A}_{j-1}$ twice. Throughout the execution of the different sub-protocols that compose our main protocol, we assume that the leader is maintaining a single tree code. The tree code is a "global variable" that retains its value between the calls to different sub-protocols. It records all the information that any protocol needs from any other protocol executed before it. Further, the leader is the only player that encodes messages using this tree code and broadcasts their encodings.

We refer to such broadcasts as 'broadcasts over the tree code'. Since only the leader makes these broadcasts, the tree code is adversary free. Suppose that at some stage in the protocol, the leader has already broadcast the encoding of the bits $b_1, \cdots, b_m$ over the tree code and wants to send the next bit b_{m+1}. To broadcast b_{m+1} over the tree code, the leader broadcasts the message $C(b_1\| \dots \|b_m\|b_{m+1})$, bit by bit (see subsection 3.4). We denote this message by simply $C(b_{m+1})$. The previous bits $b_1, \cdots, b_m$ will be clear from the context (these are all the bits communicated over the tree code so far).

4.3 Active and Passive Players

All the protocols we describe are multi-party protocols executed over a noisy broadcast network. However, we partition the players into two sets: active players and passive players. Active players are assumed to have an input and may broadcast messages during the execution of the protocol. Passive players do not have inputs and will not be broadcasting (only listening-in). Since both the types listen to the bits broadcast, they are able to compute functions of the transcripts they witnessed, and thus compute the outputs promised by the protocols. We mention that the leader will participate in any execution of any protocol as an active player.

4.4 Harmonious Executions and Fixed Length Protocols

The protocol we employ to solve the CPJ problem involves many sub-protocols that are invoked multiple times. Some of the sub-protocols that are invoked many times may involve a different set of players in each invocation. The subset of players participating in an invocation of a sub-protocol is determined adaptively, i.e., according to the players' inputs, randomness and the randomness in the channel.

Consider a sub-protocol $\mathcal{A}$ that involves k active players in any given execution, but this subset of k players might vary between two different invocations. To simplify the description and the analysis, we will describe and analyze every such protocol $\mathcal{A}$ as if players in the set $[k]$ were participating in it.

We now define a notion of a '*harmonious* invocation' that would allow us to lift our results from players in the set $[k]$ to an arbitrary subset S of participating players. Since we want the result(s) proved for the case where the players in the set $[k]$ are active (and the rest are passive) to hold in general, intuitively, we want the invocation to look similar. We require three things from a harmonious invocation. Firstly, every player in $[n]$ should know the number k of active players. Secondly, there should be a set S of size k such that a player i participates as an active player if and only if $i \in S$. Thirdly, since the k active players in the description of $\mathcal{A}$ might play different roles, we assume that the players in S know a common bijection $f : S \to [k]$ and player $i \in S$ participates as $f(i)$ in the execution of $\mathcal{A}$.

The definition above ensures the following. Firstly, every active player knows exactly who participates in the execution of $\mathcal{A}$ and in what role. This means that they can determine what parts of the transcript they receive during the execution are from which players. This allows them to 'parse' the transcript the right way. Also, since the passive players know they are passive, they stay silent throughout. This doesn't cause undesirable collisions (rounds where multiple people speak) due to players not in S interfering in the execution of the protocol. Together, these things imply that the invocation proceeds as if the players in $[k]$ were participating.

One nuance in the above discussion is in the word 'throughout'. How do the passive players know that the invocation of $\mathcal{A}$ has ended and they may need to broadcast now? All the sub-protocols we describe would require a *fixed number of broadcasts*. This number is solely determined by k and is independent of the inputs, the players' randomness or the noise in the channel. Since all the players know k, the passive players can tell when the execution ends. The fact that our protocols are of fixed length, also allows us to make sure that the rounds in which the leader is broadcasting are adversary free.

5 The OR Protocol

In this section, we design an adaptive protocol called $\texttt{OR}_n$ for the problem of computing the OR of n bits belonging to n different participants connected by an (n, ϵ)-noisy broadcast network. The protocol is given in Algorithm 3. For the rest of the section we fix n and refer to $\texttt{OR}_n$ as simply $\texttt{OR}$.

Theorem 5.1. *There exists $m \in \mathbb{N}$ and parameters $l_1, l_2, l_3 = \mathcal{O}_\epsilon(\log n)$ such that the following holds: Assume that $n \geq m$ active players with inputs $x_1, \cdots, x_n \in \{0, 1\}$, and* *any number of passive players, run the* `OR` *protocol. Then, the output* $or^{(i)}$ *of player* i *(active or passive) satisfies* $or^{(i)} = OR(x_1, \cdots, x_n) = \vee_{i\in[n]} x_i$ *with probability at least* $1 - \frac{1}{n^{20}}$*. The probability is over the noise of the channel and the randomness of the players.*

The protocol `OR` *requires a fixed number of broadcast rounds, and this number is at most* $\mathcal{O}(\log^3 n)$*.*

We note that our protocol performs better than the $\Omega(n)$ lower bound that holds for all non-adaptive protocols. The reason this is possible is because an adaptive environment allows the players to broadcast based on their input.

An Informal Discussion of the `OR` Protocol. The OR of n bits has the property that it is 1 if and only if one of the n bits is 1. If these bits belong to different participants, the participant that has 1 doesn't need to know anything about any other participant's input. All they have to do is to convince everyone that they are holding a 1. Participants holding a 0 play the complementary role. They know that their input is not going to affect the ultimate result. Hence, they might stay silent and try to hear the participants, if any, that have a 1.

Consider the following simple protocol where those participants who have a 1 (called 1-players) broadcast 1. The participants holding a 0 (called 0-players) stay silent throughout. If there is exactly one 1-player, this is the best possible protocol. After $\Theta(\log n)$ broadcasts, each participant would know that a 1-player exists with probability $1 - \frac{1}{\mathrm{poly}(n)}$. By a union bound, all n players will know the correct answer with probability $1 - \frac{1}{\mathrm{poly}(n)}$. Observe, however, that this simple algorithm fails to work if the number of 1-players is anything other than one[2].

Suppose now that the number of 1-players is at least one. In this case, if the 1-players keep broadcasting throughout, every single round of the protocol would encounter a collision and every single bit received by every participant would be adversarial. This shows that the 1-players need to coordinate *during* the protocol (since the input is worst case, no coordination is possible beforehand).

It is notable that similar situations arise in distributed networks during the so-called 'rumor-spreading' protocols. These protocols work in the environment where all participants are connected by a network unknown to any of the participants. One of the participants, call them the *leader* has a 'rumor'[3] that they want to spread to all other participants. However, if more than two neighbors of any given participant speak at the same time, the bits sent are lost. The crux of the problem is to adaptively ensure that all the participants don't keep stepping on each others' toes.

The celebrated Decay protocol [BGI92] achieves this by discounting the probability with which the players are broadcasting. In the first round of the protocol, all 1-players will

[2]It is worth mentioning that if the number of 1-players is assumed to be one always, the participants don't actually need to do anything to compute the result of the OR.

[3]This may consist of multiple bits.

broadcast 1. In each of the following $\mathcal{O}(\log n)$ rounds, each participant that broadcasted in the previous round will back-off (stay silent) randomly with probability 1/2 and with probability 1/2, broadcast 1. Once a player decides to back-off, he will never broadcast again. Thus, the number of broadcasting neighbors of any participant decreases by (roughly) 1/2 in each round. It can been proved that with constant probability, there would be a round where exactly one neighbor broadcasts, thus the message will be conveyed. Of course, the process can be repeated to boost the success rate. We follow a similar 'random-back-off' approach in our protocol.

In addition to handling the case where more than one 1-player is broadcasting, our `OR` protocol needs to also handle the case where there are no 1-players. (The assumption that there exists a 1-player is equivalent to assuming that the OR is always 1.) This introduces a complication, as in our model, if no transmissions occur, then every participant would receive an adversarial bit in every round. Since these bits might resemble legitimate broadcasts by a 1-player if they existed, it is necessary to verify that this is not the case.

Our procedure for authenticating messages is first having all the players 'sign' their messages with a unique identifier (their "name"). The length of each identifier is logarithmic in n. One of the participants, whom we call the 'leader', then verifies each message they received by broadcasting the signature. The player who signed the message can then tell the leader their bit, and the leader can, thus, verify.

Finally, note that we focused entirely on the adversary and overlooked the noise injected by the channel. We correct this by noting that if the adversary is not corrupting the transmission, correct reception by all the participants can be ensured by using error correcting codes.

5.1 The Protocol

Our description of the protocol closely follows the ideas in the foregoing section. We break our protocol into two phases: the advertising phase and the inquiry phase. The final `OR` protocol is obtained by running the advertising protocol with the given input bits, followed by an execution of the inquiry protocol.

5.1.1 Phase 1: Advertising

Phase 1 is the 'advertising' phase where the 1-players try to convey their bit to other participants. The protocol for this phase is given in Algorithm 1. Note that it is not necessary for all 1-players to convince everyone else. Even if only one 1-player is able to achieve this, the players can then convince themselves that the OR of all the bits is 1.

In our protocol, the 1-players advertise themselves by broadcasting their identity[4]. By identity we mean the player's number (that is, player i broadcasts the binary representation

[4]If convenient, it may be imagined that they also broadcast 1 multiple times and then sign it with their identity. However, since we assume that only 1-players broadcast, this step is redundant.

of i)[5]. To cope with the noise in the channel, the participants encode their identities using a constant rate error-correcting code. This guarantees that all the other participants will correctly decode with high probability.

If multiple (or zero) participants broadcast at the same time, then the bits received are adversarial. At the heart of our protocol is a method that ensures that at least one of the identities is transmit on an otherwise silent channel. We do this by making the 1-players broadcast with a probability α. If there are $1/\alpha$ participants, the number of participants that broadcast at any given time is 1 in expectation. We convert this to a high probability argument by a standard repetition technique. Note that this is similar to the random-back-off technique followed in the Decay algorithm [BGI92].

We describe our protocol using the parameters l_1, l_2, and l_3. Asymptotically, all of these parameters would grow as $\mathcal{O}_\epsilon(\log n)$. The selection of parameters is made in subsection 5.2.

Algorithm 1 `Advertising` (the first stage of the $\texttt{OR}_n$ protocol)

Input: Player $i \in [n]$ is active and his input is a bit $x_i \in \{0, 1\}$. Any number of other players may participate as passive players.

Output: The leader has $l_1 l_2$ identities $\{id_j^{(ld)}\}_{j=1}^{l_1 l_2}$ where $id_j^{(ld)} \in [n]$.

```
1: for $j_1 \leftarrow 1, 2, \cdots, l_1$ do
2:     $\alpha \leftarrow 2^{1-j_1}$.
3:     for $j_2 \leftarrow 1, 2, \cdots, l_2$ do
4:         for all players $i \in [n]$ do
5:             With probability $\alpha$,
6:             if $x_i = 1$ then
7:                 Broadcast $E_{\log_2 n, 4\epsilon}(i)$. The leader receives a corrupted codeword and decodes it to the nearest element $id^{(ld)}_{(j_1-1)l_2+j_2} \in [n]$.
8:             end if
9:         end for
10:     end for
11: end for
```

5.1.2 Phase 2: Inquiry

All that the advertising phase guarantees is that if there were 1-players, then with high probability, one of them will transmit their identity over an otherwise silent channel. It makes no claims about any of the other identities transmitted. Indeed, many of them would be adversarial. It also does not say anything about which identity was received correctly by the leader. Furthermore, if there were no 1-players, all bets are off. In this case, anything received by any participant in this phase is adversarial.

The next phase of the protocol is the inquiry phase. The protocol for this phase is given in Algorithm 2. In the protocol, the leader decodes each of the $l_1 l_2$ encodings they receive in

[5]We mention that we could have used any unique string as an 'identity'. However, we must make sure that the set of all identities is known to the leader.

the advertising phase to an identity in the set $[n]$. A fact that would turn out to be crucial in the proof is that the rounds in which the leader broadcasts the value it decoded are fixed in advance. No other participant is allowed to speak in these rounds. These 'non-adaptive' steps are also non-adversarial (see subsection 4.1). When the leader broadcasts their decoded values[6], every other participant receives only a noisy (and not adversarial) version of these broadcasts. Since this identity was encoded using an error correcting code, it will be decoded correctly by every player in the network with high probability.

Assume now that all the players decode the identity broadcast by the leader correctly. This means that all the players would now agree on a value $i \in [n]$ that was sent by the leader. This value i uniquely defines one of n active players. This player would answer the inquiry by sending their bit $\log n$ times. Moreover, no one else would be broadcasting when they do this. This implies that all the players would receive noisy (but not adversarial) copies of the bit. The players can then compute the majority bit which will be, with high probability, correct.

If there was a 1-player, one of the identities broadcast by the leader was of a 1-player. They would answer this inquiry with 1 and would be able to convince all the players that the OR is 1. If there were no 1-players, all the inquiries would be answered with 0. The players can use this information to say that the OR is 0.

For the case where all participants are 0-players, it is crucial that exactly one participant responds to each of the leader's inquiries. This is because in this case, the players say that the OR is 0 only if *all* the bits received are 0. Even if one of the responses is adversarial, the adversary can send a 1 as that response. The players would then compute value of OR (incorrectly) as 1. This is where we need the leader to know n. If the leader inquires about a string not in $[n]$, no player would respond to that inquiry making the response adversarial. Since the adversary can send a 1, in which case the output of the algorithms is 1, the algorithm will no longer be reliable.

Algorithm 2 `Inquiry` (the second stage of the $\texttt{OR}_n$ protocol)

Input: The leader has $l_1 l_2$ identities $\{id_j^{(ld)}\}_{j=1}^{l_1 l_2}$, where $id_j^{(ld)} \in [n]$. Player $i \in [n]$ is active and his input is a bit $x_i \in \{0,1\}$. Any number of other players may participate as passive players.

Output: The output of player i (active or passive) is a bit $or^{(i)}$.

```
1: for $j \leftarrow 1, 2, \cdots, l_1 l_2$ do
2:     The leader broadcasts $E_{\log_2 n, 4\epsilon}(id_j^{(ld)})$.
3:     Player $i \in [n]$ (active) decodes the leader's broadcast. If the value is $i$, then player $i$
       broadcasts the bit $x_i$ $l_3$ times.
4:     Player $i'$ (active or passive) computes the majority value, $b_j^{(i')}$, of the $l_3$ $\epsilon$-noisy copies
       of $x_i$ he received.
5: end for
6: Player $i$ computes and outputs $\vee_{j=1}^{l_1 l_2} b_j^{(i)}$.
```

[6] after re-encoding them

Algorithm 3 $\texttt{OR}_n$

```
Input: Player $i \in [n]$ is active and his input is a bit $x_i \in \{0,1\}$. Any number of other
    players may participate as passive players.
Output: The output of player $i$ (active or passive) is a bit $or^{(i)}$.
1: Players execute Advertising. The output of the leader is $\{id_j^{(ld)}\}_{j=1}^{l_1 l_2}$, $id_j^{(ld)} \in [n]$.
2: Players execute Inquiry, where the input to the leader is the list of $l_1 l_2$ identities
    $\{id_j^{(ld)}\}_{j=1}^{l_1 l_2}$.
3: Player $i$ outputs the value $or^{(i)}$ returned by the Inquiry protocol.
```

5.2 Analysis

5.2.1 The Advertising Phase

In this section we analyze the advertising stage of the protocol. Define $A \subseteq [n]$ as the set of all 1-players. That is, A is the set of players with input 1. Let $(j_1, j_2) \in [l_1] \times [l_2]$. Let $X_{j_1 j_2}$ be the set of players broadcasting in iteration (j_1, j_2) of the protocol `Advertising`. We first prove that at least one of the sets $X_{j_1 j_2}$ is singleton. This formalizes the claim made in the previous section that at least one 1-player is able to advertise non-adversarially. For this to be true, there should exist one 1-player. Thus, throughout this section, we would assume that $k = |A| \geq 1$.

Lemma 5.2. *Let $k \geq 1$ and $j_1 = \lfloor \log_2 k \rfloor + 1$. With probability at least $1 - (\frac{15}{16})^{l_2}$, there exists a $j_2 \in [l_2]$ for which $|X_{j_1 j_2}| = 1$.*

Proof. We consider two cases. We first analyze the case where $k = 1$ (and $j_1 = 1$, $\alpha = 1$). This corresponds to the simple case where there is exactly one 1-player. As discussed before, we can actually prove a stronger statement in this case. We prove that in fact *for all* $j_2 \in [l_2]$, $|X_{1 j_2}| = 1$. This is indeed the case as for any j_2, since $\alpha = 1$, the only 1-player in A transmits in every iteration $(1, j_2)$.

Now we deal with the case where $k > 1$. Since $j_1 = \lfloor \log_2 k \rfloor + 1$, $\frac{2}{k} > \alpha \geq \frac{1}{k}$. In the iteration (j_1, j_2) each of the k players in A broadcast with probability α, independently. The probability of exactly one player broadcasting in iteration (j_1, j_2) is given by

$$\Pr[|X_{j_1 j_2}| = 1] = \binom{k}{1} \alpha (1-\alpha)^{k-1} \geq (1-\alpha)^k \geq (1-\alpha)^{2/\alpha}.$$

Since we assumed that $k \geq 2$, we have $j_1 \geq 2$ and $\alpha = 2^{1-j_1} \leq 1/2$. The function $(1-x)^{1/x}$ is strictly decreasing in the range $[0,1]$. Continuing the previous equation,

$$\Pr[|X_{j_1 j_2}| = 1] \geq (1-\alpha)^{2/\alpha} \geq \left(1 - \frac{1}{2}\right)^4 = 1/16.$$

Thus, for all j_2, we have $\Pr[|X_{j_1 j_2}| \neq 1] \leq \frac{15}{16}$. Since the randomness is chosen independently for each iteration (j_1, j_2), we have

$$\Pr[\exists j_2 : |X_{j_1 j_2}| = 1] \geq 1 - \left(\frac{15}{16}\right)^{l_2}.$$

□

Theorem 5.3. *For $l_1, l_2 = \mathcal{O}(\log n)$, if $k \geq 1$, then with probability at least $1 - n^{-100}$, there exist $j_1 \in [l_1], j_2 \in [l_2]$ such that $|X_{j_1 j_2}| = 1$.*

Proof. We fix $l_1 = \lfloor \log n \rfloor + 1$ and l_2 such that $\left(\frac{15}{16}\right)^{l_2} \leq n^{-100}$. Lemma 5.2 guarantees that $\exists j_2 \in [l_2]$ such that $|X_{\lfloor \log k \rfloor + 1, j_2}| = 1$ with probability at least $1 - \left(\frac{15}{16}\right)^{l_2} \geq 1 - \frac{1}{n^{100}}$. We just set $j_1 = \lfloor \log k \rfloor + 1 \leq \lfloor \log n \rfloor + 1 = l_1$.

□

5.2.2 The Inquiry Phase

At the beginning of this stage, the leader has a list of $l_1 l_2$ identities $\{id_j\}_{j=1}^{l_1 l_2}$. This list was formed by decoding the bits received in the $(j_1, j_2)^{\text{th}}$ iteration of the advertising stage to the nearest element in $[n]$, for every $(j_1, j_2) \in [l_1] \times [l_2]$. The leader proceeds by broadcasting each of these identities and listening to the response. If all the participants are 0-players (have a 0 input), then all the responses should be 0. The leader hopes that if there exists a 1-player, one such participant would be inquired and a 1 would be received as a response. We assume throughout that the noise in the channel is less than 1/8.

We first prove that a unique (active) player responds to each of the leader's query.

Lemma 5.4. *Fix any $i \in [n]$. With probability at least $1 - n^{-29}$, all active participants decode the leader's ϵ-noisy broadcast of $E_{\log n, 4\epsilon}(i)$ correctly (see Line 3 of Algorithm 2).*

Proof. Because the rounds in which the leader broadcasts are known in advance, the others players never broadcast in these rounds, and all the players receive a noisy copy of $E_{\log n, 4\epsilon}(i)$. Lemma 3.4 implies that any given player decodes the correct value except with probability at most n^{-30}. Taking a union bound over all the n players, we get that the probability that there exists a player who decodes incorrectly is upper bounded by n^{-29}. □

Corollary 5.5. *After the leader broadcasts $E_{\log n, 4\epsilon}(i)$ for some $i \in [n]$, with probability at least $1 - n^{-29}$, exactly one player broadcasts in the next l_3 rounds (see Line 3 of Algorithm 2).*

Proof. Players only broadcast if the value they decode is their own identity. If all players decode the same value, only one player can broadcast as identities are unique. Also, since the identities are in $[n]$, at least one player will broadcast. By Lemma 5.4, this is the case with probability at least $1 - n^{-29}$. □

Having established that collisions are rare in the responses, we turn to prove that the responses would allow any participant to compute the OR correctly. For this, it is essential that the majority of l_3 broadcasts is reliable with high probability. In other words,

Lemma 5.6. *For some $l_3 = \mathcal{O}_\epsilon(\log n)$ the following holds: Suppose that each participant has l_3 (possible different) ϵ-noisy copies of a bit b (that is, each of the copies is b with probability $1-\epsilon$, independently). Then, with probability at least $1 - n^{-99}$, the majority of the l_3 copies of all players is b.*

Proof. Let $l_3 = 100c \log n$ where c is the constant given by Lemma 3.2. Lemma 3.2 says that the probability that the majority of the l_3 copies of a specific player is different than b is at most n^{-100}. A union bound over all n players shows that the assertion holds. □

Together, the two foregoing results can be interpreted as follows. Each of the $l_1 l_2$ iterations in Algorithm 2 is a query and a response. The query is the leader broadcasting $E_{\log n, 4\epsilon}(i)$ for an $i \in [n]$. The response is when (hopefully) one of the participants broadcasts their bit l_3 times. Since the rounds when the leader broadcasts are fixed in advance, all the players receive a noisy copy of $E_{\log n, 4\epsilon}(i)$. Lemma 5.4 guarantees that this is decoded to i by all the players with high probability. Since there is a unique player with identity i, exactly one player will be transmitting in the response step (Corollary 5.5). All players would receive l_3 noisy versions of this response and the majority of these l_3 responses would be correct with high probability.

We are now ready to prove the main theorem of this section:

Proof of Theorem 5.1. Let $l_1, l_2, l_3 = \mathcal{O}_\epsilon(\log n)$. The the number of broadcasts during the `OR` protocol is $l_1 l_2 (l_3 + \mathcal{O}_\epsilon(\log n)) = \mathcal{O}_\epsilon((\log n)^3)$.

We next prove the correctness of the protocol. Denote $OR(x_1, x_2, \cdots, x_n)$ by or. Note that $or = 1$ if and only if $k = |A| \geq 1$ implying it is sufficient to show that $or^{(i)} = 1$ if and only if $k = |A| \geq 1$. We break the proof into two cases.

- $\boldsymbol{k = 0}$**.** In this case, we want to show that, for any i, is holds that $or^{(i)} = 0$, with high probability. Consider an iteration j of `Inquiry` ($j \in [l_1 l_2]$). Observe that the identity $id_j^{(ld)}$ broadcast by the leader is decoded correctly by all the active players except with probability n^{-29} (Lemma 5.4). Thus, the response would be collision free (Corollary 5.5) and all players would receive l_3 noisy copies of $x_{id_j^{(ld)}}$. Lemma 5.6 guarantees that the majority of the l_3 copies of any player i, $b_j^{(i)}$, would be $x_{id_j^{(ld)}} = 0$, except with probability at most n^{-99}. Coupling the above statements, for any player i, it holds that $b_j^{(i)} = 0$, with probability at least $1 - 2n^{-29}$. A union bound over all $l_1 l_2$ possible values for j gives that $or^{(i)} = 0$ for each i except with probability $\frac{2 l_1 l_2}{n^{29}} \leq \frac{1}{n^{25}}$ for large enough n.

- $\boldsymbol{k \geq 1}$**.** In this case, we want to show that $or^{(i)} = 1$ for every i with high probability. Since $or^{(i)} = \vee_{j=1}^{l_1 l_2} b_j^{(i)}$, it is sufficient to show that there exists a j such that $b_j^{(i)} = 1$ for all i with high probability. Since $k \geq 1$, we can apply Theorem 5.3 to say the $\exists j_1, j_2 : |X_{j_1 j_2}| = 1$ with probability at least $1 - n^{-100}$. This means that in the $(j_1, j_2)^{\text{th}}$ iteration of Algorithm 1, exactly one 1-player, say $i' \in A$, broadcast their identity. The leader thus receives a noisy version of $E_{\log n,4\epsilon}(i')$ which was decoded correctly with probability at least $1-n^{-30}$ (Lemma 3.4). If this happens, the leader will broadcast $E_{\log n,4\epsilon}(i')$ in the j^{th} iteration of Algorithm 2 where $j=(j_1-1)l_2+j_2$. By Corollary 5.5, with probability at least $1-n^{-29}$, player i' would broadcast $x_{i'}=1$ in response, and will be the only player broadcasting. His broadcast will be repeated l_3 times ensuring that all other players hear a majority of the responses correctly with probability at least $1-n^{-99}$ (Lemma 5.6). Thus, for every i it holds that $b_j^{(i)}=1$ with probability at least $1-n^{-100}-n^{-30}-n^{-29}-n^{-99}\geq 1-n^{-20}$ for large n, and the assertion follows.

□

6 The Longest Common Prefix Protocol

In the *Longest Common Prefix* (LCP) problem, n players are each holding a private input $v_i\in\{0,1\}^{\leq n}$. The goal is computing the longest prefix $l(v_1,\cdots,v_n)$ common to all the strings in $v_1,\cdots,v_n$ (see definition in section 3).

In this section, we design an adaptive protocol called $\texttt{LCP}_n$, that computes the LCP of n bit strings belonging to n different participants connected by an (n,ϵ)-noisy broadcast network. Let $m\in\mathbb{N}$ be the constant from Theorem 5.1, and assume without loss of generality that $m\geq 2^{20}$. The protocol $\texttt{LCP}_n$ for $n\geq m$ is given in Algorithm 4. The protocol $\texttt{LCP}_n$ for $n<m$ is simple: Each player broadcasts each of his (at most m) input bits m^{1000} times. Then, players compute the LCP of the majorities by themselves.

Theorem 6.1. *There exists a constant $c_1\in\mathbb{N}$ such that the following holds: Assume that n active players with inputs $v_1,\cdots,v_n\in\{0,1\}^{\leq n}$, a leader with input $v\in\{0,1\}^{\leq n}$, and any number of passive players, run the $\texttt{LCP}$ protocol. Then, the output $lcp^{(i)}$ of player i (active or passive) satisfies $lcp^{(i)}=|l(v_1,\cdots,v_n,v)|$ with probability at least $1-\min\{n^{-9},2^{-9}\}$. The probability is over the noise of the channel and the randomness of the players.*

The protocol $\texttt{LCP}$ requires a fixed number of broadcast rounds, and this number is at most $\mathcal{O}_\epsilon(\log^4 n)$.

Theorem 6.1 clearly holds for $n<m$. For the rest of the section we will assume that $n\geq m$.

Applications of LCP. We first note that the OR function can be formulated as an LCP problem. In the case of OR, each of n players has a private input bit x_i that can be either 0 or 1 and the players want to know if there exists someone who has the bit 1. Each of these bits can be viewed as a string v_i of length 1. In this case, $l(v_1,\cdots,v_n)$ has three possible values. It can either be the empty string ε, the string 1, or the string 0. If $l(v_1,\cdots,v_n)$ is the empty string, it means that some of the v_i are 1 and others are 0. The OR in this case would be 1. If $l(v_1, \cdots, v_n)$ is 1, it means that all the v_i are 1 and the OR is 1. Finally, if the LCP is 0, all the strings are 0 and so is the OR.

The LCP framework can also be used as a general purpose 'progress-check' subroutine in many scenarios. An example might be the identity problem. This problem considers n players, each holding a private input bit x_i. The players communicate towards the common goal of each knowing all n input bits. Consider an adaptive protocol that runs in the following way. All the players i maintain a belief x_{ij} about the private input bit of every other player j. These beliefs are random at the beginning of the protocol. As the players communicate, they transmit information about their input bits to the other participants who can then update their beliefs. We show how LCP can be used to quantify the progress made in such a scenario. Each participant i forms a string $v_i = x_{i1}x_{i2} \cdots x_{in}$. This string represents their current belief about the bits of all the other players. The 'diagonal' entries in the v_i are the correct input bits, *i.e.* $v_{ii} = x_i$. Suppose $|l(v_1, \cdots, v_n)| = l$. This means that $v_{1j} = v_{2j} = \cdots = v_{nj}$ for all $1 \leq j \leq l$. In particular, $v_{ij} = v_{jj} = x_j$ for all $i \in [n], j \in [l]$. In words, the first l players have successfully conveyed their bits to the other players. The output of the LCP protocol tells the players that they can now only focus on the rest of the $n - l$ bits[7].

We are interested in solving the CPJ problem, which subsumes the identity problem (see subsection 2.2). In this problem, each player i has a private function $f_i : \{0,1\}^{i-1} \to \{0,1\}$. Let $x_1 = f_1(\varepsilon)$, $x_2 = f_2(x_1)$, $x_3 = f_3(x_1x_2)$ and so on. The players mutual goal is computing the string $X = x_1x_2 \cdots x_n$. Again, players i might have a belief $v_i = v_{i1}v_{i2} \cdots v_{in}$ about the *entire* string X. Since all the players know their private function, we can assume $f_i(v_i[1 : i-1]) = v_{ii}, \forall i$. If $l(V)$ has length l, the first l bits of the belief of all the n players match. Since every player i sets v_{ii} to be correct given the prefix preceding it, this also means that these l bits match the first l bits in X and the players only need to compute the remaining bits.

An Informal Description of the LCP Protocol. Having highlighted the importance of the LCP problem, we now describe an efficient broadcast protocol to compute the LCP. The protocol finds the longest common prefix by performing a binary search over all possible prefix lengths, *i.e.* 0 to n. In order to check whether a prefix of length l is common to all the strings, player 1 (the leader) broadcasts $\mathcal{O}(\log n)$ hashes of $v_1[1 : l]$. This number is enough to ensure that except with polynomially small probability, any player j with $v_j[1 : l] \neq v_1[1 : l]$ will have a different set of hashes. These hashes are broadcasted multiple times to ensure correct reception (repetition code). A more efficient version of the protocol with constant rate codes can be described. However, we persist with the less efficient version because of its simplicity and the fact that broadcasting hashes is not the bottleneck in the broadcast complexity of our protocol.

Once all the players know the hashes of the leader's string, they can compute the

[7]This is possible only if the model is adaptive. Indeed, this is how we would use LCP in our protocols.

corresponding hashes of their own string and see if the two sets match. If the strings are the same in the first l bits, then the hashes would match. For two strings that differ in their first l bits, the hashes should be different. The players who find an inconsistency in any of the hashes assume a bit 1 and those who have the same set of hashes assume the bit 0. Let the bit assumed by player i be called b_i. If any of the b_i is 1, one of the players doesn't agree with the set of hashes broadcast and thus, on the prefix of length l. If all the b_i are 0, all the players (hopefully) agree on the prefix of length l. By running the `OR` protocol with inputs $b_1, \cdots, b_n$, all players will know if $|l(V)|$ is at least l or not. In a binary search framework, this is sufficient to nail down $l(V)$.

6.1 The Protocol

We formalize the ideas described. Every player i has a string v_i of length n. The set of all the n strings is V. The players together implement a binary search framework to compute $|l(V)|$. Once this length is known to all the players, the players can themselves compute $l(V) = v_i[1 : |l(V)|]$. Since $l(V)$ is common to all strings, the expression on the right is independent of i, as it should be.

All the n players operate the binary search framework in parallel. They execute the same protocol with independent 'private' variables. We exercise care in our description of the protocol and denote the private variable x of player i with the superscript $x^{(i)}$. In our description below, we use n_0 to denote the smallest integer such that $2^{n_0} > n$. We initialize our range for $|l(V)|$ to $[0, 2^{n_0}-1] \supseteq [0, n]$. This helps avoiding inconsistencies due to rounding and in turn, keeping the number of broadcasts in our protocol independent of the output. Protocols that are of a predetermined length are easier to handle as a subroutine because one doesn't have to worry about players not knowing if the subroutine has finished execution (see subsection 4.4).

Let $H_k : \{0,1\}^* \to \{0,1\}^k$ be a (probabilistic) hash function satisfying $\forall x \neq y \in \{0,1\}^*$ it holds that $H_k(x) = H_k(y)$ with probability 2^{-k}. The protocol will use $H = H_{20 \log n}$.

6.2 Analysis

We fix c_1 to be 100 times the constant in Lemma 3.2. At a very high level, all the players in the protocol described try to find out $|l(V)|$. This value is all they need in order to compute $l(V)$. They do so by maintaining a range $[beg^{(i)}, end^{(i)}]$ containing the value $|l(V)|$. After every iteration of the while loop in line 3, the range $[beg^{(i)}, end^{(i)}]$ shrinks by 1/2. Lemma 6.7 formalizes these invariants. Before we prove that, however, we wish to write a few technical lemmas concerning our protocol.

Fact 6.2. *The number of broadcast in any iteration of the while loop in* `LCP` *is fixed based on n. It is independent of the players' inputs, randomness, or the noise in the channel.*

Lemma 6.3. *All players (active and passive) run the same number of iteration of the while loop and finish the execution of the protocol* `LCP` *after a fixed number of broadcast rounds.*

Algorithm 4 The $\texttt{LCP}_n$ Protocol

Input: Player $i \in [n]$ is active and his input is a string $v_i \in \{0,1\}^{\leq n}$. The leader has a string $v \in \{0,1\}^{\leq n}$. Any number of other players may participate as passive players.

Output: The output of player i (active or passive) is an integer $lcp^{(i)} \in \mathbb{N}$.

```
1: $beg^{(i)} \leftarrow 0$.
2: $end^{(i)} \leftarrow 2^{n_0} - 1$.
3: **while** $beg^{(i)} < end^{(i)}$ **do**
4:     $mid^{(i)} \leftarrow \left\lceil \frac{beg^{(i)}+end^{(i)}}{2} \right\rceil$.
5:     $H^{(i)} \leftarrow H(v_i(beg^{(i)} : mid^{(i)}])$.
6:     The leader broadcasts $H^{(ld)}$, $c_1 \log n$ times.
7:     Each player $i$ receives $c_1 \log n$ noisy copies of $H^{(ld)}$ and computes their majority $H_{ld}^{(i)}$.
       (The $j^{th}$ bit of $H_{ld}^{(i)}$ is obtained by taking the majority of the $j^{th}$ bits of the $c_1 \log n$ noisy
       copies received by player $i$.)
8:     $b^{(i)} \leftarrow (H_{ld}^{(i)} \neq H^{(i)})$.
9:     Players execute $\texttt{OR}_n$. Player $i \in [n]$ are active and have inputs $b^{(1)}, b^{(2)}, \cdots, b^{(n)}$. All
       other players are passive. The output of player $i$ (active or passive) is $or^{(i)}$.
10:    **if** $or^{(i)} = 0$ **then**
11:        $beg^{(i)} \leftarrow mid^{(i)}$
12:    **else**
13:        $end^{(i)} \leftarrow mid^{(i)} - 1$
14:    **end if**
15: **end while**
16: Player $i$ outputs $beg^{(i)}$.
```

Proof. To show this, we show that the following holds before the j^{th} iteration of the while loop for all players i.

$$end^{(i)} - beg^{(i)} = 2^{n_0+1-j} - 1.$$

We prove this using induction on j. For $j = 1$, the invariant holds trivially. Suppose the invariants hold till before iteration j. The j^{th} iteration would only take place if $n_0 \geq j$. In the j^{th} iteration, player i computes $mid^{(i)} = \lceil \frac{beg^{(i)}+end^{(i)}}{2} \rceil = \lceil beg^{(i)} + \frac{2^{n_0+1-j}-1}{2} \rceil = beg^{(i)} + 2^{n_0-j}$ and either sets $beg^{(i)}$ to $mid^{(i)}$ or $end^{(i)}$ to $mid^{(i)} - 1$. We denote the values of beg and end after this iteration using primes. In the first case, $end'^{(i)} - beg'^{(i)} = end^{(i)} - mid^{(i)} = 2^{n_0+1-j} - 1 - 2^{n_0-j} = 2^{n_0-j} - 1$. In the second case, $end'^{(i)} - beg'^{(i)} = mid^{(i)} - 1 - beg^{(i)} = 2^{n_0-j} - 1$.

Thus, all the players take exactly n_0 iterations to finish execution. Since the number of broadcasts in any iteration is fixed (Fact 6.2), all the players finish execution after the same number of broadcasts. □

Claim 6.4. *All the invocations of* `OR` *are harmonious and all the leader's broadcasts are adversary-free.*

Proof. The total number (Lemma 6.3) and the size (Fact 6.2) of the iterations are pre-determined. In each iteration, the communication rounds in which the leader is broadcasting during the execution of the protocol are pre-determined, and do not depend on players' inputs, their randomness or the noise in the channel.

By Lemma 6.3, all players invoke `OR` the same number of times. Since all players participate in each invocation of `OR`, all invocations are harmonious. □

Lemma 6.5. *For any player $i \in [n]$ in any iteration of the while loop, $H_{ld}^{(i)} = H^{(ld)}$ except with probability at most $\frac{20 \log n}{n^{100}}$.*

Proof. Since the leader's broadcasts are adversary-free (Claim 6.4), all players i receive $c_1 \log n$ noisy copies of each of the leaders hashes. Lemma 3.2, for a coordinate $j \in [20 \log n]$, the probability that the j^{th} coordinate of $H_{ld}^{(i)}$ is different from the j^{th} coordinate of $H^{(ld)}$, is at most $\frac{1}{n^{100}}$. Taking a union bound over the $20 \log n$ coordinates, we get that $H_{ld}^{(i)} = H^{(ld)}$ except with probability at most $\frac{20 \log n}{n^{100}}$. □

Lemma 6.6. *For any player $i \in [n]$ and any iteration of the while loop, $v_i(beg^{(i)} : mid^{(i)}] \neq v_{ld}(beg^{(ld)} : mid^{(ld)}]$ if and only if $b^{(i)} = 1$, except with probability at most $\frac{1}{n^{18}}$.*

Proof. Note that $b^{(i)} = 1$ if and only if $H_{ld}^{(i)} \neq H^{(i)}$ and break the proof into the following two cases.

- $\boldsymbol{v_i(beg^{(i)} : mid^{(i)}] = v_{ld}(beg^{(ld)} : mid^{(ld)}]}$**.** In this case, we want to prove that $b^{(i)} = 0$ with high probability. Since $v_i(beg^{(i)} : mid^{(i)}] = v_{ld}(beg^{(ld)} : mid^{(ld)}]$, we have $H^{(i)} = H^{(ld)}$. Lemma 6.5 implies that $H_{ld}^{(i)} = H^{(ld)} = H^{(i)}$ except with probability at most $\frac{20 \log n}{n^{100}}$. Thus, $b^{(i)} = 0$ with probability at least $1 - \frac{20 \log n}{n^{100}}$.

- $\boldsymbol{v_i(beg^{(i)} : mid^{(i)}] \neq v_{ld}(beg^{(ld)} : mid^{(ld)}]}$**.** In this case, we want to prove that $b^{(i)} = 1$ with high probability. $H^{(i)}$ and $H^{(ld)}$ consist of $20 \log n$ independent hashes of two different strings. Thus, the probability that $H^{(i)} \neq H^{(ld)}$ is $1 - \frac{1}{n^{20}}$. Again, Lemma 6.5 gives us $H^{(i)}_{ld} = H^{(ld)} \neq H^{(i)}$ with probability at least $1 - \frac{1}{n^{20}} - \frac{20 \log n}{n^{100}} \geq 1 - \frac{1}{n^{18}}$ (recall that we assume $n \geq m$). Thus, $b^{(i)} = 1$ with the same probability.

Combining the two cases above gives the result. □

Let $V = \{v_1, \cdots, v_n\}$. Lemma 6.6 says that the bits $b^{(i)}$ correctly capture whether or not player i agrees with the leader on the coordinates in the range $(beg^{(i)} : mid^{(i)}]$. If any one of the $b^{(i)}$ is 1, then not all the strings in V agree on these coordinates. In turn, this says that $|l(V)|$ has to be strictly less that $mid^{(i)}$. On the other hand, if all the $b^{(i)}$ are 0, all the strings match on these coordinates and $|l(V)|$ is at least $mid^{(i)}$. Thus, $l(V)$ either contains all the coordinates $(beg^{(i)} : mid^{(i)}]$ or none of the coordinates $(mid^{(i)} : end^{(i)}]$. Our update of the variables $beg^{(i)}$ and $end^{(i)}$ maintains the following invariant.

Lemma 6.7. *With probability at least $1 - \frac{1}{n^{15}}$ over the noise in the channel and the players randomness, the following invariant holds before every iteration of the while loop: For $i \in [n]$, all $beg^{(i)}$ have a common value beg and all $end^{(i)}$ have a common value end such that $beg \leq |l(V)| \leq end$.*

Furthermore, for any passive player $i \notin [n]$, the value of $(beg^{(i)}, end^{(i)}) = (beg, end)$ in all the iterations except with probability at most $1 - \frac{1}{n^{15}}$.

Proof. We proceed via induction on the iteration number. The invariant trivially holds before the first execution of the loop as $2^{n_0} - 1 \geq n$, by definition. Suppose it holds before the j^{th} iteration.

Consider the j^{th} iteration. Let E_i be the event that $v_i(beg^{(i)} : mid^{(i)}] \neq v_{ld}(beg^{(ld)} : mid^{(ld)}]$. Lemma 6.6 says that $b^{(i)} = 1$ if and only if E_i occurs, except with probability at most $\frac{1}{n^{18}}$. Taking a union bound over all the n active players, we get that for all $i \in [n]$, $b^{(i)} = 1$ if and only if E_i occurs, except with probability at most $\frac{1}{n^{17}}$. Therefore, except probability at most $\frac{1}{n^{17}}$, it holds that $\vee_{i=1}^{n} b^{(i)} = 1$ if and only if $\exists i \in [n]$ such that E_i occurs.

Since all the invocations of `OR` are harmonious (Claim 6.4), we can apply Theorem 5.1 to conclude that for any $i \in [n]$, $or^{(i)} = \vee_{i=1}^{n} b^{(i)}$ except with probability at most $\frac{1}{n^{20}}$. We union bound over all the n active players to get that, except with probability at most $\frac{1}{n^{19}}$, it holds that $\forall i \in [n]$, $or^{(i)} = \vee_{i=1}^{n} b^{(i)}$

Combining the two results, we get with probability at least $1 - \frac{1}{n^{17}} - \frac{1}{n^{19}} \geq 1 - \frac{1}{n^{16}}$, all the $or^{(i)}$ are the same value or' and or' is 1 if and only if $\exists i \in [n]$ such that E_i occurs. Since the values of $or^{(i)}$ are the same for all i, the players update $beg^{(i)}$ and $end^{(i)}$ identically and the first part of the invariant is maintained.

For the second part of the invariant, note that if $\exists i \in [n]$ such that E_i occurs, then $|l(V)| < mid$, where mid is the common value of $mid^{(i)}$ (observe that since $mid^{(i)} = \left\lceil \frac{beg^{(i)} + end^{(i)}}{2} \right\rceil$, and since $beg^{(i)}$ and $end^{(i)}$ have common values, so do $mid^{(i)}$). Thus, setting end to $mid - 1$

preserves the invariant ($or' = 1$). Similarly, if $v_i(beg^{(i)} : mid^{(i)}]$ is the same for all i and $|l(V)| \geq beg$, we have $|l(V)| \geq mid$ and setting beg to mid preserves the invariant ($or' = 0$).

We observe that the while loop is executed at most $1 + \log n$ times. Thus, the total error probability is bounded by $\frac{\log n}{n^{16}} \leq \frac{1}{n^{15}}$.

For the last part, we note that (Theorem 5.1) any passive player has the correct result of any execution of the `OR` protocol except with probability $1 - \frac{1}{n^{20}}$. If a passive player i has the correct result of all the $\log n + 1$ invocations of `OR`, their updates to $beg^{(i)}$ and $end^{(i)}$ will match the updates made by the active players. Thus, their final values of $beg^{(i)}$ and $end^{(i)}$ will also be the same. The probability that player i has the correct result for $\log n + 1$ executions is, by a union bound, at least $1 - \frac{1+\log n}{n^{20}} \geq 1 - \frac{1}{n^{15}}$. □

Lemma 6.8. *When the loop defined at Line 3 ends for any player i (active or passive), $beg^{(i)} = end^{(i)}$.*

Proof. In every execution of the loop, $beg^{(i)} < end^{(i)}$ which implies $mid^{(i)} > beg^{(i)}$ and $end^{(i)} \geq mid^{(i)}$. Let $beg'^{(i)}$ and $end'^{(i)}$ be the values of $beg^{(i)}$ and $end^{(i)}$ after an iteration ends. It is sufficient to prove that $beg'^{(i)} \leq end'^{(i)}$. This follows because either $beg'^{(i)} = beg^{(i)}$ and $end'^{(i)} = mid^{(i)} - 1$ or $beg'^{(i)} = mid^{(i)}$ and $end'^{(i)} = end^{(i)}$. In the first case, $beg'^{(i)} = beg^{(i)} < mid^{(i)} = end'^{(i)} + 1$. In the second, $beg'^{(i)} = mid^{(i)} \leq end^{(i)} = end'^{(i)}$. □

Proof of Theorem 6.1. The complexity of the `LCP` protocol is $\mathcal{O}_\epsilon(\log^4 n)$, as the while loop is executed at most $1 + \log n$ times, and in each iteration, the `OR` protocol is called (see Theorem 5.1). The broadcasts in the `OR` protocol constitute the dominant term in the number of broadcasts.

With probability at least $1 - \frac{1}{n^{15}}$, the invariants in Lemma 6.7 hold before every iteration of the while loop. This fact along with Lemma 6.8 implies that the output $lcp^{(i)}$ (which is set to $beg^{(i)}$) of player i (active or passive) satisfies $lcp^{(i)} = |l(v_1, \cdots, v_n)|$ with probability at least $1 - \frac{1}{n^{15}}$.

For the passive players, Lemma 6.7 says that any such player i has the same (correct) value of $lcp^{(i)}$ as the active players except with probability at most $\frac{1}{n^{15}}$. This completes the proof. □

7 Protocol for Correlated Pointer Jumping

This section contains our main result - a linear adaptive broadcast protocol for solving the correlated pointer jumping (CPJ) problem for n players. In the CPJ setting, every player $i \in [n]$ has a private function $f_i : \{0,1\}^{i-1} \to \{0,1\}$. For $i = 1$, this is just a constant which we denote using $f_1(\varepsilon)$ (here, ε is the empty string). The functions can be 'composed' to define strings $\sigma_i \in \{0,1\}^i$ as follows,

$$\begin{aligned} \sigma_0 &= \varepsilon \\ \sigma_{i+1} &= \sigma_i \| f_{i+1}(\sigma_i), \forall i \in [n]. \end{aligned} \tag{1}$$

(Recall that '$\|$' denotes the string concatenation operator). We define $\mathrm{CPJ}(f_1, \cdots, f_n)$ to be the string σ_n defined above.

7.1 Informal Discussion

Consider the following (optimal) protocol for solving the CPJ problem when the network is noise free: In round i, player i computes and broadcasts the value $f_i(\sigma_{i-1})$, where σ_{i-1} is the transcript received so far. In other words, player 1 knows his function f_1 and can thus compute σ_1 on its own. After computing, it broadcasts σ_1 to all other players. This then allows player 2 to compute σ_2. Player 2 can then broadcast σ_2. The process continues until the last player computes and broadcasts σ_n. Since σ_{i+1} is just σ_i plus an additional bit, player 2 in the protocol above doesn't need to transmit both the bits of σ_2. Only the last bit would suffice. The same argument can be applied to all the players implying that the number of broadcasts in this protocol is exactly n.

If we want to simulate this protocol in a noisy environment, the first thing that comes to mind is that each bit can be repeated enough times so that a majority of the copies received by every player is correct with high probability. Since the total number of players is n, we would need to repeat each bit roughly $\log n$ times. We are then guaranteed that the protocol would compute σ_n correctly except with small (inverse polynomial in n) error probability.

At first, this appears to be the best one can hope for. Indeed, every player i has to know the bit (the function output) of all the players $j < i$ in order to compute their bit. Thus, $n - 1$ players need to know σ_1, $n - 2$ players need to know σ_2 and so on. This implies the first $n/2$ bits need to be known by at least $n/2$ players. The first $n/2$ players would thus have to repeat their bit $\Theta(\log n)$ times and the $\log n$ blowup cannot be avoided.

The chink in this armor is that all the players do not need to know the bits immediately after they are broadcast. If i is large, player i has to listen to a lot of people before getting a chance to speak. At first, this doesn't seem to give any leverage. This is because the i transmissions that happen before player i gets a chance to speak may be completely unrelated. What we want is to somehow have each subsequent transmission reinforce the confidence in all the previous transmissions.

This is exactly where tree codes come into the picture. The idea behind this construction is to have an error-correcting code that can be computed online. Theorem 3.7 says that for *any* string of length k broadcast on the tree code, the probability that a player with a noisy version of the encoding of these k bits, decodes a suffix of length l incorrectly is exponentially small in l.

With this new insight, one can consider the following protocol. Player 1 broadcasts their bit σ_1 once (or any constant number of times). However, this time they do this by encoding their bit over a tree code and broadcasting the encoding. Theorem 3.6 promises that there exist tree codes for which this encoding is constant-length. With constant probability, player 2 would decode this correctly. After they do that, they can send the next bit in σ_2 over the tree code. Player 3 would then decode *both* these bits correctly with the *same* constant probability. They can then send $f_3(\sigma_2)$ over the tree code. This process can go on till all the n players have broadcasted their inputs.

Note that each of the $\mathcal{O}(n)$ transmissions in this protocol can be wrong with constant probability. The probability that this protocol goes through is thus exponentially small in n. One way to avoid this is to periodically check whether the bits transmitted so far were correct. If the transmissions are correct, we can carry on with the rest of the protocol. Otherwise, the best thing to do is to repeat the part that hasn't been received correctly. In other words, such a protocol should have (at least) two types of broadcasts, regular broadcasts (regular rounds) and check broadcasts (check rounds). The regular broadcasts would serve to compute σ_i for higher and higher i's while the check rounds would make sure that our computation this far is reliable. Furthermore, we would want that the check rounds don't involve too many broadcasts. Our aim is to implement *all* the check rounds using $\mathcal{O}(n)$ broadcasts.

We prove that such a protocol actually works. In what follows, we recursively define a sequence $\{\mathcal{A}_j\}_{j\geq 0}$ of protocols. The protocol $\mathcal{A}_j$ would compute the string σ_{2^j}. At a *very* high level, the protocol $\mathcal{A}_{j+1}$ would first run $\mathcal{A}_j$ to compute σ_{2^j}. Assuming that this string is known to 2^{j+1} players, it would then invoke $\mathcal{A}_j$ again to compute the last 2^j bits in $\sigma_{2^{j+1}}$. These two executions are identical because assuming all players in $[2^{j+1}]$ know σ_{2^j}, player 2^j+1 can compute $f_{2^j+1}(\sigma_{2^j})$ without any extra information (just like player 1 in the first execution). Similarly, player 2^j+i only needs to know the bits output by the $i-1$ players $[2^j+1,\cdots,2^j+i-1]$ to compute $f_{2^j+i}(\sigma_{2^j+i-1})$ (exactly like player i in the first execution). Thus, this procedure is just $\mathcal{A}_j$ run with an offset (or better, a 'history string') of length 2^j. After these two executions, it would check which portions were received correctly. We defer more details about $\mathcal{A}_j$ to the future sections.

7.2 The Protocol $\mathcal{A}_0$

In Algorithm 5, we describe $\mathcal{A}_0$, the first protocol in our recursive sequence. This protocol serves to compute one step in the induction described in Equation 1.

To compute and broadcast the first string σ_1, only player 1 interacts with the leader. Player 1 broadcasts $\sigma_1 = f_1(\varepsilon)$. Let $b^{(ld)}$ be the noisy copy of σ_1 received the leader. The leader broadcasts $b^{(ld)}$ over the tree code. Herein lies a very important feature of our protocol. On the face of it, the idea of the leader repeating the first player's transmission over the tree code doesn't seem very productive. Why not just have player 1 broadcast over the tree code directly? The answer is that making the leader repeat the broadcast allows us to make the tree-code adversary free. Throughout our CPJ protocol, it will only be the leader who would broadcast on the tree code and furthermore, they would do so in predetermined rounds (see subsection 4.1). All the other players would be silent in these rounds and receive a noisy copy of the leaders broadcast. This would set the stage up for Theorem 3.7 which requires players to receive a noisy version of the bits sent on the tree code.

There is a subtlety here. It is totally possible that the bit the leader sent on the tree code was adversarial. This can happen, for instance, when the round before the leader transmitted encountered a collision and the bit received by everyone (including the leader's bit $b^{(ld)}$) in this round was adversarial. When we say that the tree code is adversary free, all we mean is that this (possibly adversarial) bit was broadcast non-adversarially. This means that (almost) all the other players would decode the tree code to the same adversarial bit. This way of avoiding the situation where different players get different adversarial bits in the same broadcast round, with only a constant factor slowdown, might be of independent interest.

After the leader's transmission, player 1 and the leader check, using the $\texttt{LCP} = \texttt{LCP}_1$ protocol, whether the leader's reception was correct. If so, the leader broadcasts the bit 1 over the tree code. In the other case, where the leader's reception was flipped, the leader broadcasts the bit 0. This bit broadcast, b_2, tells the other players whether or not the first bit broadcast was reliable[8]

We now describe the protocol (see Algorithm 5). There are two participants, player 1 and the leader. Player 1 has a bit (a constant function) that he communicates to the leader. The leader then repeats this bit over the tree code. This is followed by a call to LCP where player 1 and the leader check if the bit was sent by player 1 was received correctly. The output of this call to LCP is broadcast by the leader over the tree code. The encoder for the tree code used by the leader is denoted by C (see subsection 4.2), and the decoder used by the players to decode the relevant information from the tree code is denoted by DECODE (see subsection 7.4).

Algorithm 5 $\mathcal{A}_0$

```
Input: Player 1 is active and his input is a function $f_1 : \{0,1\}^0 \to \{0,1\}$. Players $i \in [n]\backslash\{1\}$
    are passive.
Output: The output of player $i \in [n]$ is a string $s^{(i)}$, where $|s^{(i)}| \leq 1$.
1: Player 1 broadcasts $b = f(\varepsilon)$. The leader receives $b^{(ld)}$.
2: The leader broadcasts $C(b^{(ld)})$.
3: Player 1 and the leader run $\texttt{LCP}_1$ with inputs $b$ and $b^{(ld)}$ (respectively). Denote the
    leader's output by $lcp^{(ld)}$.
4: The leader broadcasts a padded version of his output $C(lcp^{(ld)} \| 0^{c_0+1})$.
5: Each player $i \in [n]$ computes and outputs $\texttt{DECODE}^{(i)}(0)$.
```

7.3 The Protocol $\mathcal{A}_j$

The protocol $\mathcal{A}_j$ involves 2^j players other than leader. It is designed to compute (up-to) 2^j steps of the induction described in Equation 1.

[8]The LCP in this case would only offer a constant probability of success which could have been achieved by repeating the broadcasts a constant number of times. We, however, keep the LCP version because it mirrors the subsequent protocols we will describe.

The protocol $\mathcal{A}_j$ does this using two successive invocations of $\mathcal{A}_{j-1}$ followed by a check round, implemented using $\mathtt{LCP} = \mathtt{LCP}_{2^j}$. The first invocation of $\mathcal{A}_{j-1}$ would compute the first 2^{j-1} steps of the induction. The output of this invocation would be a string $s_1^{(i)}$ for each player i that records the steps that were computed correctly. Thus, $|s_1^{(i)}| \le 2^{j-1}$.

All the players $i \in [2^j]$ believe that the bits in $s_1^{(i)}$ were computed correctly. Accordingly, they proceed to compute the rest of the bits. This is done using a second invocation of $\mathcal{A}_{j-1}$. Because the i^{th} player assumed that $s_1^{(i)}$ was correct, they would participate as player $i - |s_1^{(i)}|$ in this invocation. We denote player i's view of the output of this invocation by $s_2^{(i)}$. As for the first invocation, $|s_2^{(i)}| \le 2^{j-1}$.

Up until now, all the players assumed that the values of $s_1^{(i)}$ and $s_2^{(i)}$ they computed were correct. Since the channel is noisy, this will not always hold. To ensure that the strings they computed are indeed correct, players use the protocol LCP with the strings $s_1^{(i)} \| s_2^{(i)}$ as input. The leader's output for this invocation, $lcp^{(ld)}$, is broadcast over the tree code. $lcp^{(ld)}$ gives the length of the prefix that was computed correctly in the two invocation of $\mathcal{A}_{j-1}$. Since $lcp^{(ld)}$ is broadcast over the tree code by the leader, the players run DECODE to compute the final output.

Observe that the role of player i in the second execution depends on their output $s_1^{(i)}$ of the first execution. Since $s_1^{(i)}$ is correct only with high probability (and not all the time), the second invocation of $\mathcal{A}_{j-1}$ may not be harmonious in every run of the protocol $\mathcal{A}_j$. We choose our parameter c_0 to ensure that the second invocation of $\mathcal{A}_{j-1}$ is harmonious a large fraction of the times. For all other times, we rely on LCP to detect and correct the errors introduced. The invocation of LCP is always harmonious.

The protocol $\mathcal{A}_j$ is described in Algorithm 6.

7.4 Parsing the Tree Code

We motivated why using tree codes in our protocol might be a good idea. In this section, we focus on how participants parse the information broadcasted by the leader over the tree code. This has two parts. Firstly, the participants are required to decode the path $\in \Sigma^*$ on the tree code to the nearest bit string $\in \{0,1\}^*$. We empathies that the players decode all the messages they received from the leader since the beginning of the execution of the protocol. The decoding is done using the standard Hamming-distance based decoders. We assume throughout this section that this preprocessing has already been performed, and that the player have access to some underlying decoded string. The more important part is to 'parse' the decoder's output to get 'information'.

We now turn to the second part. Recall that during the execution of $\mathcal{A}_0$, the leader first broadcasts $b^{(ld)}$, the value he received when player 1 broadcast b. He then broadcasts $lcp^{(ld)}$[9], the result of the LCP execution on b and $b^{(ld)}$. This value is either 0 or 1, where 1 indicates that $b^{(ld)} = b$ with high probability, while a 0 values means that the reception was

[9]The leader also pads the LCP result with a constant number of bits. This is not relevant to the discussion here.

Algorithm 6 $\mathcal{A}_j$

```
Input: Player $i \in [2^j]$ is active and his input is a function $f_i : \{0,1\}^{i-1} \to \{0,1\}$. Player
    $i \in [n] \setminus [2^j]$ are passive.
Output: The output of player $i \in [n]$ is a string $s^{(i)}$ of length at most $2^j$.
 1: Players execute $\mathcal{A}_{j-1}$ for the first time: Player $i \in [2^{j-1}]$ participates as active player $i$
    with input $f_i$. All other players participate as a passive players. Denote the output of
    the execution for player $i \in [n]$ by $s_1^{(i)}$.
 2: Players execute $\mathcal{A}_{j-1}$ for the second time: Player $i \in [2^j]$ checks if $i \in \left[|s_1^{(i)}|+1, |s_1^{(i)}|+2^{j-1}\right]$.
    If so, he participates as the active player $i$ with input function
    $g_i : \{0,1\}^{i-1-|s_1^{(i)}|} \to \{0,1\}$, where $g_i(t) = f_i(s_1^{(i)} \| t)$. Otherwise, participates as a passive
    player. Denote the output of the execution for player $i \in [n]$ by $s_2^{(i)}$.
 3: Player $i \in [2^j] \cup \{ld\}$ sets $v_i$ to $s_1^{(i)} \| s_2^{(i)}$, but replaces coordinate $i$ of $s_1^{(i)} \| s_2^{(i)}$ by the value
    $f_i(v_i[1 : i-1])$ if $\left|s_1^{(i)} \| s_2^{(i)}\right| \geq i$.
 4: Players execute $\texttt{LCP}_{2^j}$: Player $i \in [2^j]$ participates as active player $i$ with input $v_i$. The
    leader is participating with the input $v^{(ld)}$. All other layers participate as a passive
    players. Denote the output of the execution for player $i \in [n]$ by $lcp^{(i)}$.
 5: The leader broadcasts a padded version of their output $C\left(lcp^{(ld)} \| 0^{(c_0+1)(j+1)}\right)$.
 6: Player $i \in [n]$ computes and outputs $\texttt{DECODE}^{(i)}(j)$.
```

faulty. Both $b^{(ld)}$ and $lcp^{(ld)}$ are broadcast over the tree code. Let $i \in [n]$. Assume that the decoded value of $b^{(ld)}$ by player i is b', and that the decoded value of $lcp^{(ld)}$ by player i is lcp'. The function $\texttt{DECODE}^{(i)}(0)$ is computed by player i (alone), as follows: If $lcp' = 1$ output b'. Otherwise output ε (the empty string).

Recall that during the execution of the protocol $\mathcal{A}_j$, the protocol $\mathcal{A}_{j-1}$ is called twice, and in each of these executions, the leader is broadcasting messages over the tree code. Then, the LCP protocol is executed to get the LCP of the strings $s_1^{(i)} \| s_2^{(i)}$ ($i \in [2^j]$). The leader broadcasts his result of the LCP execution, $lcp^{(ld)}$, over the tree code. Since $lcp^{(ld)} \in [2^j]$, we may assume that $lcp^{(ld)}$ is written over the tree code as a bit string of length exactly j.

Let $i \in [n]$. Since during the executions of $\mathcal{A}_{j-1}$, a fixed number of bits are written over the tree code, we may view the underlying transcript decoded from the tree code by player i as composed of three parts: The first two, T_1 and T_2, consist of the values decoded by player i of the messages that the leader broadcasts over the tree code during the first and second execution of $\mathcal{A}_{j-1}$ (respectively). The third part, $lcp^{(i)}$, is the decoded value of $lcp^{(ld)}$ by player i. The function $\texttt{DECODE}^{(i)}(j)$ is computed by player i (alone), as suggested by Algorithm 7. We also write this using the equation:

$$\texttt{DECODE}^{(i)}(j) = \left(\texttt{DECODE}^{(i)}(j-1)_1 \| \texttt{DECODE}^{(i)}(j-1)_2\right)[1 : lcp^{(i)}] \tag{2}$$

Algorithm 7 $\texttt{DECODE}^{(i)}(j)$

```
1: Run $\texttt{DECODE}^{(i)}(j-1)$ where the underlying string decoded from the tree code is $T_1$ to
   get the output string $r_1$.
2: Run $\texttt{DECODE}^{(i)}(j-1)$ where the underlying string decoded from the tree code is $T_2$ to
   get the output string $r_2$.
3: Output $(r_1 \| r_2)[1 : lcp^{(i)}]$.
```

8 Analysis of the CPJ Protocol

We first claim that the length of an execution of each of the protocols $\mathcal{A}_j$ is fixed.

Lemma 8.1. *The protocol $\mathcal{A}_j$ ($j \geq 0$) require a fixed number of broadcasts, regardless of whether or not it is executed harmoniously.*

Proof. First recall that an invocation of $\texttt{LCP}_m$ requires a fixed number of broadcasts for every $m \in \mathbb{N}$ (Lemma 6.3). Observe that this holds even if the invocation is not harmonious. Thus, if players start an invocation of $\texttt{LCP}_m$ at the same time, they will end their execution together. The protocol $\mathcal{A}_0$ consists of constant number of broadcasts, as well as an execution of $\texttt{LCP}_1$. Thus, $\mathcal{A}_0$ is of fixed length. For $j \geq 1$, the protocol $\mathcal{A}_j$ is two successive invocations to $\mathcal{A}_{j-1}$ followed an execution of $\texttt{LCP}_{2^j}$ and by the leader broadcasting the $\texttt{LCP}$ output (requiring $j+1$ bits) and $(c_0+1)(j+1)$ bits of padding over the tree code. Since these numbers are fixed, if we assume that $\mathcal{A}_{j-1}$ is fixed length, then so is the length of $\mathcal{A}_j$. Thus, we can show by induction on j that the executions of $\mathcal{A}_j$ are of a fixed number of rounds. □

Using Lemma 8.1, it can be shown that during the execution of the protocol $\mathcal{A}_j$ ($j \geq 0$), the leader is broadcasting in pre-determined rounds. All players know these rounds in advance and will not be broadcasting. Thus, these rounds are adversary free (see subsection 4.1).

Corollary 8.2. *During the execution of the protocol $\mathcal{A}_j$ ($j \geq 0$), the rounds in which the leader is broadcasting are adversary free.*

Let Σ be the set of labels for the edges in our tree code. We assume, without loss of generality, that $\Sigma = \{0,1\}^{\varsigma}$ for some $\varsigma \in \mathbb{N}$. At any stage of the execution of the protocol $\mathcal{A}_j$ ($j \geq 0$), the leader would have broadcast the encoding of a sequence of k bits over the tree code, for some k (see subsection 4.2). Saying it differently, the leader broadcasts the labels on k edges on some path in the tree code. Denote these list of edges broadcast by the leader by $\pi^{(ld)} \in \Sigma^k = \{0,1\}^{k\varsigma}$. Corollary 8.2 implies that each player receives a noisy copy of the edges. Let $\pi^{(i)} \in \{0,1\}^{k\varsigma} = \Sigma^k$ be the copy of $\pi^{(ld)}$ received by player $i \in [n]$. Let D be the Hamming distance based decoder for the tree code defined in subsection 3.4, so that $D(\pi^{(i)})$ is a string in $\{0,1\}^*$.

Lemma 8.3. *At any point in the execution of the protocol $\mathcal{A}_j$ ($j \geq 0$), for any player $i \in [n]$, if $\pi^{(ld)}, \pi^{(i)} \in \Sigma^k$, then for any $k_0 \in \mathbb{N}$,*

$$\Pr\left[\left|l(D(\pi^{(ld)}), D(\pi^{(i)}))\right| \leq k - k_0\right] \leq K_1 \exp(-K_2 k_0),$$

for some constants K_1 and K_2.

Proof. We assume that the encoding function C used in our protocol defines a $(2, \alpha = \frac{1}{2}, \Sigma)$-tree code for a constant size Σ. Such a tree code exists by Theorem 3.6. Thus, each transmission on the tree code is of a constant size. Denote this size by ς. Suppose the noise in the channel is less than $1/4\varsigma$, so that the probability that one symbol on the tree code is corrupted by noise is at most $1/4 = \alpha/2$. This is without loss of generality, as it can be ensured by repeating every communicated bit constant number of times.

Since the leader knows their own broadcast, $\pi^{(ld)}$ is noise-free. This implies that $\pi^{(ld)} = \overline{C}(D(\pi^{(ld)}))$ (recall Definition 3.5). By Corollary 8.2, we also know that $\pi^{(i)}$ is a δ-noisy version of $\pi^{(ld)}$ with $\delta < \alpha/2$. The assertion follows for applying Theorem 3.7 with $s = D(\pi^{(ld)})$. □

We analyze protocols $\mathcal{A}_j$ sequentially in the following subsections. The high level idea is to use the analysis of $\mathcal{A}_{j-1}$ twice to get bounds for $\mathcal{A}_j$. We will prove that with high probability both the calls to $\mathcal{A}_{j-1}$ within $\mathcal{A}_j$ are harmonious, and that harmonious executions result in large expected progress. Thus, the progress roughly doubles at every step to remain within a constant factor of the number of transmissions.

8.1 Analyzing $\mathcal{A}_0$

In this section, we assume that the execution of $\mathcal{A}_0$ is harmonious. Since $\mathcal{A}_0$ involves only one player other than the leader, the definition of harmonious reduces to exactly one player broadcasting in line 1 of the description. Of course, this player will also participate in the execution of $\texttt{LCP}_1$ which will be harmonious.

Lemma 8.4 (Progress)**.** *With probability at least $1 - \epsilon - \frac{1}{2^9}$, a harmonious execution of $\mathcal{A}_0$ will have $b^{(ld)} = b$ and $lcp^{(ld)} = 1$, where the probability is over the noise of the channel and the randomness of the players.*

Proof. Since the execution is harmonious, there no collision in line 1 and the bit received by the leader is just an ϵ-noisy copy of b. The leader's noisy version $b^{(ld)}$ is the same as b except with probability ϵ. If indeed $b^{(ld)} = b$, then $l(b, b^{(ld)}) = b$. Apply Theorem 6.1 to conclude that $\Pr[lcp^{(ld)} = 1] \geq 1 - \frac{1}{2^9}$ when $l(b, b^{(ld)}) = b$. However, $l(b, b^{(ld)}) = b$ happens with probability at least $1 - \epsilon$. Thus, $\Pr[lcp^{(ld)} = 1] \geq 1 - \frac{1}{2^9} - \epsilon$ □

We measure the progress made by our algorithm using the variable $lcp^{(ld)}$. The foregoing lemma shows that the expected progress made by a harmonious invocation of $\mathcal{A}_0$ is large. We now prove that this progress is 'correct' with high probability. Our notion of correctness is that strings $s^{(i)}$ output by player i when $\mathcal{A}_0$ ends, satisfies $s^{(i)} = \sigma_{|s^{(i)}|}$ (see Equation 1).

Lemma 8.5 (Correctness)**.** *There exists a constant $c_0 \in \mathbb{N}$ such that the following holds: For any player $i \in [n]$, at the end of a harmonious execution of $\mathcal{A}_0$, it holds that $|s^{(i)}| = lcp^{(ld)}$ and $s^{(i)} = \sigma_{|s^{(i)}|}$, with probability at least $1 - \frac{1}{2^8}$, where the probability is over the noise of the channel and the randomness of the players.*

Proof. Consider the messages $C(b^{(ld)})$ and $C(lcp^{(ld)})$ broadcast by the leader over the tree code. The players decode after these messages are broadcast. The probability that all the bits that are not a part of the padding are decoded correctly is at least $1 - K_1 \exp(-K_2(c_0+1))$ by Lemma 8.3. We can set c_0 to a constant value large enough so that $K_1 \exp(-K_2(c_0+1)) \leq 1/2^{10}$. For the rest of this proof, we condition on the event E that the decoding of non-padding bits by player i is correct.

When E occurs, the output of the $\texttt{DECODE}^{(i)}(0)$ procedure is $s^{(i)} = b^{(ld)}[1 : lcp^{(ld)}]$ (here we view $b^{(ld)}$ as a string of length 1). If $lcp^{(ld)} = 0$, player i sets $s^{(i)} = \varepsilon$. Recall that $\sigma_0 = \varepsilon$, thus $s^{(i)} = \sigma_0$. Now assume that $lcp^{(ld)} = 1$. Since the $\texttt{LCP}$ invocation was harmonious, by Theorem 6.1, its output is incorrect with probability at most $1/2^9$. Then, with probability at least $1 - \frac{1}{2^9} - \Pr(\bar{E})$, we have $\sigma_1 = b = b^{(ld)} = s^{(i)}$, and the assertion follows. □

8.2 Analyzing $\mathcal{A}_j$

Our goal in this section is to prove analogues of Lemma 8.4 and Lemma 8.5 for $\mathcal{A}_j$, $j \geq 1$. We would rely on the following reasoning. Consider a harmonious execution of $\mathcal{A}_1$. This would involve two players, say player 1 and 2, as well as the leader. Player 1 (and the leader) will first run $\mathcal{A}_0$ harmoniously. The output $s_1^{(i)}$ of this execution will be of length $lcp_1^{(ld)}$ and be correct for both the players with high probability. We assume for now that this execution is indeed correct for both players. If the outputs $s_1^{(i)}$ are of the same length for $i \in [2]$, then second execution would be harmonious and we can re-apply Lemma 8.5 to conclude that the output of the second execution is also correct with high probability. If both the outputs are correct, the subsequent $\texttt{LCP}$ should output the concatenation of the two outputs. Thus, the length of the (correct) output would double at every step to stay larger than a constant fraction of the number of players. We extend this reasoning to general $\mathcal{A}_j$ in the following lemmas.

Lemma 8.6 (Progress)**.** *There exists a constant $c_0 \in \mathbb{N}$ such that the following holds: In a harmonious execution of $\mathcal{A}_j$, for $j \leq \lfloor \log n \rfloor$, we have*

$$\mathbb{E}\left[lcp^{(ld)}\right] \geq 2^j \left(1 - \frac{1}{2^9} - \epsilon - 6\sum_{i=1}^{j} 2^{-7i}\right).$$

Lemma 8.7 (Correctness)**.** *There exists a constant $c_0 \in \mathbb{N}$ such that the following holds: For any player $i \in [n]$, at the end of a harmonious execution of $\mathcal{A}_j$, for $j \leq \lfloor \log n \rfloor$, it holds that $|s^{(i)}| = lcp^{(ld)}$ and $s^{(i)} = \sigma_{|s^{(i)}|}$, with probability at least $1 - 2^{-8j-8}$, where the probability is over the noise of the channel and the randomness of the players.*

We prove Lemma 8.6 and Lemma 8.7 in subsubsection 8.2.1. The proof of Theorem 1.1 follows from these lemmas, as well as from the fact that the protocol $\mathcal{A}_j$ requires at most $\mathcal{O}(2^j)$ broadcast rounds.

8.2.1 Proof of Lemma 8.6 and Lemma 8.7

In this section we next prove Lemma 8.6 and Lemma 8.7 together via induction on j. The base case ($j = 0$) is given by Lemma 8.4 and Lemma 8.5 respectively. We assume both the results for $j - 1$ and reason about $\mathcal{A}_j$.

$\mathcal{A}_j$ begins with a call to $\mathcal{A}_{j-1}$ on the first half of the players. Since these players are pre-specified, this call is harmonious and, by the induction hypothesis (correctness), both of the following holds for any $i \in [n]$, except with probability $1 - 2^{-8j}$,

$$|s_1^{(i)}| = lcp_1^{(ld)}, \tag{3}$$

$$s_1^{(i)} = \sigma_{|s_1^{(i)}|}. \tag{4}$$

Here $lcp_1^{(ld)}$ denotes the value of $lcp^{(ld)}$ in this execution of $\mathcal{A}_{j-1}$. Let E_1 be the event that $s_1^{(i)} = \sigma_{lcp_1^{(ld)}} = \sigma'$ for all $i \in [2^j]$. By union bound over Equation 3 and Equation 4 for all players in $[2^j]$, we get that $\Pr[E_1] \geq 1 - 2^{-7j}$.

Claim 8.8. *Assuming that E_1 occurs, the second execution of $\mathcal{A}_{j-1}$ is harmonious.*

Proof. We will only use the fact that the strings $s_1^{(i)}$ are of the same length, for all $i \in [2^j]$. In the second execution of $\mathcal{A}_{j-1}$, if $i \in \left[|s_1^{(i)}| + 1, |s_1^{(i)}| + 2^{j-1}\right]$, then player i takes the role of active player $i - |s_1^{(i)}|$. Since $|s_1^{(i)}| \leq 2^{j-1}$, it holds that $\left[|s_1^{(i)}| + 1, |s_1^{(i)}| + 2^{j-1}\right] \subseteq [2^j]$. Since $|s_1^{(i)}| = |\sigma'|$ for every $i \in [2^j]$, it holds that every player $i \in [|\sigma'| + 1, |\sigma'| + 2^{j-1}]$ takes the role of player $i - |\sigma'|$. Since the function $i \to i - |\sigma'|$ is a bijection between the set of active players participating and the set of roles, the second execution of $\mathcal{A}_{j-1}$ is harmonious. (Note that all players $i \in [n] \setminus [2^j]$ will always participate in this execution $\mathcal{A}_{j-1}$ as passive players.) □

Assume that E_1 occurs. The second execution of $\mathcal{A}_{j-1}$ involves the functions g_i. The functions g_i were defined in a way so that their domain is consistent with the role of player i. Since this execution is also harmonious, we again apply the induction hypothesis (correctness) to conclude that for any $i \in [n]$, except with probability 2^{-8j},

$$|s_2^{(i)}| = lcp_2^{(ld)}, \tag{5}$$

$$s_2^{(i)} = \tau_{|s_2^{(i)}|}. \tag{6}$$

Here $lcp_2^{(ld)}$ denotes the value of $lcp^{(ld)}$ in this (second) execution of $\mathcal{A}_{j-1}$, and $\tau_{|s_2^{(i)}|}$ is the string obtained by replacing all f_i by g_i in Equation 1. Let E_2 be the event that

$s_2^{(i)} = \tau_{lcp_2^{(ld)}} = \tau'$ for all $i \in [2^j]$. By union bound over Equation 4 for all players in $[2^j]$, we get that $\Pr[E_2|E_1] \geq 1 - 2^{-7j}$. If we denote $\sigma' \| \tau'$ by ν, we get

Claim 8.9. *Assuming that E_1 and E_2 occur, it holds that $\nu = \sigma_{|\nu|}$.*

Proof. Since $\sigma' = \sigma_{|\sigma'|}$ (definition of E_1 and Equation 4), it is sufficient to show that the i^{th} bit of τ' is the same as the i^{th} bit of $\sigma_{|\nu|}[|\sigma'| + 1 : |\nu|]$. Or, in other words, the i^{th} bit of τ' is the $(|\sigma'| + i)^{\text{th}}$ bits of $\sigma_{|\nu|}$. We prove this by induction on i. The statement holds for $i = 0$. Assume it holds for all values up to $i - 1$. The i^{th} bit of $\tau' = \tau_{lcp_2^{(ld)}}$ is $g_{|\sigma'|+i}(\tau'[1 : i-1]) = f_{|\sigma'|+i}(\sigma' \| (\tau'[1 : i-1]))$ by definition. By the induction hypothesis, this is the same as $f_{|\sigma'|+i}\left(\sigma' \| (\sigma_{|\nu|}[|\sigma'| + 1 : |\sigma'| + i - 1])\right) = f_{|\sigma'|+i}(\sigma_{|\nu|}[1 : |\sigma'| + i - 1])$ which is the $(|\sigma'| + i)^{\text{th}}$ bit of $\sigma_{|\nu|}$. □

Assume that E_1 and E_2 occur. Then, $s_1^{(i)} \| s_2^{(i)} = \sigma_{|\nu|}$ for all the players $i \in [2^j]$, and coordinate i of $s_1^{(i)} \| s_2^{(i)}$ is $f_i(v_i[1 : i-1])$. Therefore, Line 3 sets v_i to $v_i = s_1^{(i)} \| s_2^{(i)} = \sigma_{|\nu|}$ for all the players $i \in [2^j]$ (as the replacing part of Line 3 doesn't have any effect). Since $l(v_1, \cdots, v_{2^j}) = |\nu| = lcp_1^{(ld)} + lcp_2^{(ld)}$, and since the $\mathtt{LCP}_{2^j}$ call is harmonious, Theorem 6.1 implies that $lcp^{(ld)} \geq lcp_1^{(ld)} + lcp_2^{(ld)}$ except with probability at most 2^{-9j}. Let E_3 be the event that $lcp^{(ld)} \geq lcp_1^{(ld)} + lcp_2^{(ld)}$. We have $\Pr(E_3 \mid E_1, E_2) \geq 1 - 2^{-9j}$. The following claim proves Lemma 8.6.

Claim 8.10.

$$\mathbb{E}[lcp^{(ld)}] \geq 2^j \left(1 - \frac{1}{2^9} - \epsilon - 6\sum_{i=1}^{j} 2^{-7i}\right).$$

Proof. We first condition on the events E_1, E_2, E_3 to use the bounds for $lcp_1^{(ld)}$ and $lcp_2^{(ld)}$ give by the induction hypothesis.

$$\begin{aligned}
\mathbb{E}[lcp^{(ld)}] &\geq \Pr[E_1, E_2, E_3] \cdot \mathbb{E}[lcp^{(ld)} \mid E_1, E_2, E_3] + (1 - \Pr[E_1, E_2, E_3]) \cdot 0 \\
&\geq (1 - 2 \cdot 2^{-7j} - 2^{-9j}) \cdot \mathbb{E}[lcp^{(ld)} \mid E_1, E_2, E_3] \\
&\geq (1 - 3 \cdot 2^{-7j}) \cdot \left(\mathbb{E}[lcp_1^{(ld)} + lcp_2^{(ld)} \mid E_1, E_2, E_3]\right).
\end{aligned}$$

Next we prove bounds on $\mathbb{E}[lcp_1^{(ld)} \mid E_1, E_2, E_3]$ and $\mathbb{E}[lcp_2^{(ld)} \mid E_1, E_2, E_3]$:

$$\begin{aligned}
\mathbb{E}[lcp_1^{(ld)} \mid E_1, E_2, E_3] &\geq \mathbb{E}[lcp_1^{(ld)}] - \mathbb{E}[lcp_1^{(ld)} \mid \bar{E}_1 \vee \bar{E}_2 \vee \bar{E}_3] \Pr(\bar{E}_1 \vee \bar{E}_2 \vee \bar{E}_3) \\
&\geq \mathbb{E}[lcp_1^{(ld)}] - 2^{j-1} \Pr(\bar{E}_1 \vee \bar{E}_2 \vee \bar{E}_3) \\
&\geq \mathbb{E}[lcp_1^{(ld)}] - 2^{j-1} \left(\Pr(\bar{E}_1) + \Pr(\bar{E}_2 \mid E_1) + \Pr(\bar{E}_3 \mid E_1, E_2)\right) \\
&\geq \mathbb{E}[lcp_1^{(ld)}] - 3 \cdot 2^{-6j-1}.
\end{aligned}$$

Also,

$$\mathbb{E}[lcp_2^{(ld)} \mid E_1, E_2, E_3] \geq \mathbb{E}[lcp_2^{(ld)} \mid E_1] - \mathbb{E}[lcp_2^{(ld)} \mid E_1, \bar{E}_2 \vee \bar{E}_3] \Pr(\bar{E}_2 \vee \bar{E}_3 \mid E_1)$$

$$\begin{aligned}
&\geq \mathbb{E}[lcp_2^{(ld)} \mid E_1] - 2^{j-1}\Pr(\bar{E}_2 \vee \bar{E}_3 \mid E_1)\\
&\geq \mathbb{E}[lcp_2^{(ld)} \mid E_1] - 2^{j-1}\left(\Pr(\bar{E}_2 \mid E_1) + \Pr(\bar{E}_3 \mid E_1, E_2)\right)\\
&\geq \mathbb{E}[lcp_2^{(ld)} \mid E_1] - 2 \cdot 2^{-6j-1}.
\end{aligned}$$

Using these bounds and the induction hypothesis, we get

$$\begin{aligned}
\mathbb{E}[lcp^{(ld)}] &\geq (1 - 3\cdot 2^{-7j}) \cdot \left(\mathbb{E}[lcp_1^{(ld)} + lcp_2^{(ld)} \mid E_1, E_2, E_3]\right)\\
&\geq (1 - 3\cdot 2^{-7j}) \cdot \left(\mathbb{E}[lcp_1^{(ld)}] + \mathbb{E}[lcp_2^{(ld)} \mid E_1] - 5\cdot 2^{-6j-1}\right)\\
&\geq (1 - 3\cdot 2^{-7j}) \cdot \left(2^j\left(1 - \frac{1}{2^9} - \epsilon - 6\sum_{i=1}^{j-1} 2^{-7i}\right) - 5\cdot 2^{-6j-1}\right)\\
&\geq 2^j\left(1 - \frac{1}{2^9} - \epsilon - 6\sum_{i=1}^{j} 2^{-7i}\right).
\end{aligned}$$

□

For Lemma 8.7, note that the input to the `LCP` call in Algorithm 6 are the strings v_i calculated in Line 3. These strings satisfy the property that their i^{th} bit is equal to f_i applied to their first $i-1$ bits. The following claim proves that strings that satisfy this property have a 'correct' LCP.

Claim 8.11. *If $V = \{v_i\}_{i\in[m]}$ is a set of strings that satisfies $v_i[i] = f_i(v_i[1 : i-1])$ for every $i \in [|l(V)|]$, then $l(V) = \sigma_{|l(V)|}$.*

Proof. We prove by induction on i that the first $i \leq |l(V)|$ bits of $l(V)$ and $\sigma_{|l(V)|}$ match. The 0^{th} bit matches trivially. Assume all bits up to $i-1$ match and consider bit $i \leq |l(V)|$. Since $l(V)$ is a prefix of v_i, the i^{th} bit of $l(V)$ is the same as the i^{th} bit of v_i. But, $v_i[i] = f_i(v_i[1 : i-1]) = f_i(\sigma_{i-1})$ by the induction hypothesis. By definition, $f_i(\sigma_{i-1}) = \sigma_i[i]$, and the assertion follows. □

Recall that the leader's input to the $\texttt{LCP}_{2^j}$ execution is $v^{(ld)} = s_1^{(ld)} \| s_2^{(ld)}$. Let $V = \{v_1, \cdots, v_n, v\}$. By Theorem 6.1 and Claim 8.11, except with probability at most 2^{-9j}, the result of the $\texttt{LCP}_{2^j}$ execution, $lcp^{(ld)}$, satisfies

$$v^{(ld)}[1 : lcp^{(ld)}] = l(V) = \sigma_{|l(V)|} = \sigma_{lcp^{(ld)}}.$$

Fix a player $i \in [n]$. We show that player i's output is correct and of length $lcp^{(ld)}$, by showing that it matches the output of the leader with high probability. First, note that the leader's output $s^{(ld)}$ satisfies

$$\begin{aligned} s^{(ld)} &= \mathtt{DECODE}^{(ld)}(j) \\ &= \left(\mathtt{DECODE}^{(ld)}(j-1)_1 \| \mathtt{DECODE}^{(ld)}(j-1)_2\right)[1 : lcp^{(ld)}] \quad \text{(by Equation 2)} \\ &= \left(s_1^{(ld)} \| s_2^{(ld)}\right)[1 : lcp^{(ld)}] \\ &= v^{(ld)}[1 : lcp^{(ld)}] \\ &= \sigma_{lcp^{(ld)}} = \sigma_{|s^{(ld)}|} \end{aligned} \tag{7}$$

We finish the proof of Lemma 8.7 by proving that the output $s^{(i)}$ of any player i, is the same as $s^{(ld)}$ with high probability.

Claim 8.12. *For any player i, the output $s^{(i)} = s^{(ld)}$ except with probability at most 2^{-8j-9}.*

Proof. The leader broadcasts $lcp^{(ld)}$ on the tree code with $(c_0+1)(j+1)$ bits of padding. For any player i, the result of $\mathtt{DECODE}^{(i)}(j)$ will differ from the result of the leader only if the size of the incorrect suffix is more than the size of the padding. We upper bound this probability using Lemma 8.3 by $K_1 \exp(-K_2(c_0+1)(j+1)) \le 2^{-8j-9}$ for a sufficiently large c_0. □

By a union bound, except with probability at most $2^{-9j} + 2^{-8j-9} \le 2^{-8j-8}$, both Equation 7 and Claim 8.12 hold. Note that the inequality holds for $j > 9$. For smaller j, we just repeat LCP a constant number of times to force the result. This finishes the proof.

References

[ABE+16] Noga Alon, Mark Braverman, Klim Efremenko, Ran Gelles, and Bernhard Haeupler. Reliable communication over highly connected noisy networks. In *PODC*, pages 165–173, 2016. 4

[BE14] Mark Braverman and Klim Efremenko. List and unique coding for interactive communication in the presence of adversarial noise. In *Proceedings of the IEEE Symposium on Foundations of Computer Science*, FOCS, pages 236–245, 2014. 4

[BEGH16] Mark Braverman, Klim Efremenko, Ran Gelles, and Bernhard Haeupler. Constant-rate coding for multiparty interactive communication is impossible. In *STOC*, pages 999–1010, 2016. 4

[BGI92] Reuven Bar-Yehuda, Oded Goldreich, and Alon Itai. On the time-complexity of broadcast in multi-hop radio networks: An exponential gap between determinism and randomization. *J. Comput. Syst. Sci.*, 45(1):104–126, 1992. 3, 8, 15, 17

[BGMO16] Mark Braverman, Ran Gelles, Jieming Mao, and Rafail Ostrovsky. Coding for interactive communication correcting insertions and deletions. In *ICALP*, pages 61:1–61:14, 2016. 4

[BKN14] Zvika Brakerski, Yael Tauman Kalai, and Moni Naor. Fast interactive coding against adversarial noise. *J. ACM*, 61(6):35:1–35:30, 2014. 4

[BR11] Mark Braverman and Anup Rao. Towards coding for maximum errors in interactive communication. In *STOC*, pages 159–166, 2011. 4, 5, 10

[Bra12] Mark Braverman. Towards deterministic tree code constructions. In *ITCS*, pages 161–167, 2012. 4

[EGH15] Klim Efremenko, Ran Gelles, and Bernhard Haeupler. Maximal noise in interactive communication over erasure channels and channels with feedback. In *ITCS*, pages 11–20, 2015. 4

[FK00] Uriel Feige and Joe Kilian. Finding OR in a noisy broadcast network. *Inf. Process. Lett.*, 73(1-2):69–75, 2000. 4

[Gal88] Robert G. Gallager. Finding parity in a simple broadcast network. *IEEE Transactions on Information Theory*, 34(2):176–180, 1988. 2, 3

[Gam87] Abbas El Gamal. Open problems presented at the 1984 workshop on specific problems in communication and computation sponsored by bell communication research. *Appeared in "Open Problems in Communication and Computation", by Thomas M. Cover and B. Gopinath (editors). Springer-Verlag*, 1987. 2

[GH15] Ran Gelles and Bernhard Haeupler. Capacity of interactive communication over erasure channels and channels with feedback. In *SODA*, pages 1296–1311, 2015. 4

[GHK+16] Ran Gelles, Bernhard Haeupler, Gillat Kol, Noga Ron-Zewi, and Avi Wigderson. Towards optimal deterministic coding for interactive communication. *SODA*, 2016. 4

[GHS14] Mohsen Ghaffari, Bernhard Haeupler, and Madhu Sudan. Optimal error rates for interactive coding I: Adaptivity and other settings. In *STOC*, pages 794–803, 2014. 4

[GKS08] Navin Goyal, Guy Kindler, and Michael Saks. Lower bounds for the noisy broadcast problem. *SIAM Journal on Computing*, 37(6):1806–1841, 2008. 2, 3, 4, 10

[GMS11] Ran Gelles, Ankur Moitra, and Amit Sahai. Efficient and explicit coding for interactive communication. In *FOCS*, pages 768–777, 2011. 4

[GMS14] Ran Gelles, Ankur Moitra, and Amit Sahai. Efficient coding for interactive communication. *IEEE Transactions on Information Theory*, 60(3):1899–1913, 2014. 4

[Hae14] Bernhard Haeupler. Interactive channel capacity revisited. In *FOCS*, pages 226–235, 2014. 4

[KM98] Eyal Kushilevitz and Yishay Mansour. An $\omega(d \log(n/d))$ lower bound for broadcast in radio networks. *SIAM J. Comput.*, 27(3):702–712, 1998. 4

[KR13] Gillat Kol and Ran Raz. Interactive channel capacity. In *STOC*, pages 715–724, 2013. 4

[MS14] Cristopher Moore and Leonard J. Schulman. Tree codes and a conjecture on exponential sums. In *ITCS*, pages 145–154, 2014. 4

[New04] Ilan Newman. Computing in fault tolerance broadcast networks. In *CCC*, pages 113–122, 2004. 4

[Pel07] David Peleg. Time-efficient broadcasting in radio networks: A review. In *Distributed Computing and Internet Technology ICDCIT*, pages 1–18, 2007. 3

[RS94] Sridhar Rajagopalan and Leonard J. Schulman. A coding theorem for distributed computation. In *STOC*, pages 790–799, 1994. 4

[Sch92] Leonard J. Schulman. Communication on noisy channels: A coding theorem for computation. In *FOCS*, pages 724–733, 1992. 4

[Sch93] Leonard J. Schulman. Deterministic coding for interactive communication. In *STOC*, pages 747–756, 1993. 4, 10

[Sch96] Leonard J. Schulman. Coding for interactive communication. *IEEE Transactions on Information Theory*, 42(6):1745–1756, 1996. 4, 11