Coding for Interactive Communication: A Survey

Ran Gelles
Faculty of Engineering, Bar-Ilan University
ran.gelles@biu.ac.il

Version 1.3, October 2019

Acknowledgments

I would like to thank Mark Braverman, Keren Censor-Hillel, Bernhard Haeupler, Jieming Mao, Anup Rao, and Leonard Schulman for invaluable comments on this manuscript and for insightful discussions. I would also like to thank Omer Dayan, Batya Karp, Ronen Voloshin, and Mary Wootters for spotting errors in previous versions. I would like to thank Mark also for clarifying and simplifying the tree code construction in Section 3.1.1 and for his continuous support. This work is supported in part by NSF grant No. CCF-1149888, and in part by the Israel Science Foundation (ISF) through grant No. 1078/17.

Revision History:
Version 1.0 (Oct, 2015) — draft for comments
Version 1.2 (Oct, 2017) — FnT version
Version 1.3 (Oct, 2019) — correcting some typos and clarifying several vague statements.

Contents

1 Introduction **1**
- 1.1 Motivation . . . 1
- 1.2 The setting . . . 2
- 1.3 Parameters that we care about . . . 3
- 1.4 Parameters that make a difference . . . 4
- 1.5 Organization . . . 5

2 Noise Resilience **7**
- 2.1 The tree codes paradigm . . . 7
- 2.2 The rewind-if-error paradigm . . . 16
- 2.3 Upper bounds on the maximal noise . . . 20

3 The Hunt for Efficient Constructions **23**
- 3.1 Efficient schemes for random noise with reduced parameters . . . 23
- 3.2 Efficient coding schemes over BSC: Potent tree codes . . . 28
- 3.3 Efficient coding schemes for adversarial noise: Suboptimal noise resilience . . . 30
- 3.4 Efficient coding schemes for adversarial noise: Optimal noise resilience . . . 32

4 Adaptive Coding Schemes **38**
- 4.1 Adaptive coding schemes: The speak-or-listen model . . . 38
- 4.2 Adaptive coding schemes: The adaptive-termination and the speak-at-will models . . . 41

5 Communication-Efficient Coding Schemes **45**
- 5.1 Rate $1 - O(\sqrt{\varepsilon \log 1/\varepsilon})$ for random noise . . . 46
- 5.2 Rate $1 - O(\sqrt{\varepsilon})$ for random noise . . . 48
- 5.3 Rate upper bounds and the order of speaking . . . 52

6 Coding Schemes over Different Noisy Channels **55**
6.1 Channels with noiseless feedback 55
6.2 Erasure channels 67
6.3 Channels with insertions and deletions 70
6.4 Quantum channels 71

7 Multiparty Interactive Communication **73**
7.1 Coding schemes for networks with random noise 74
7.2 Coding for networks with adversarial noise: Upper bounds on the maximal noise 79
7.3 Coding schemes for networks with adversarial noise: Synchronous setting 79
7.4 Coding schemes for networks with adversarial noise: Asynchronous setting 80

8 Applications and Related Topics **83**
8.1 Noise-resilient formulas 83
8.2 Noise-resilient private protocols (do not exist) 86
8.3 Noise-resilient interactive proofs 88
8.4 Noise-resilient perpetual (one-way) communication 89

A Addendum **92**
A.1 The Hunt for Efficient Constructions (Section 3): 92
A.2 Communication-Efficient Coding Schemes (Section 5) 93
A.3 Coding Schemes over Different Noisy Channels (Section 6): 93
A.4 Multiparty Interactive Communication (Section 7): 93

B Summary of Known Schemes **95**

References **99**

1

Introduction

1.1 Motivation

Assume Alice and Bob play chess with each other. Since they live in different countries, they play over the phone—every evening Alice calls Bob and they communicate the next move. Now assume the phone line is noisy. While Bob declares his next move "`B2 to B4`", Alice hears "`D2 to D4`". Many days later, when Bob declares a victory, Alice rejects his claims: on her board Bob is not even close to being victorious.

This situation—where two parties *interact* with each other over a noisy communication channel—is the topic of this manuscript. As opposed to the standard error-correction setting in which one side has some information to convey to the other side, here both sides need to convey information to each other. One could let the parties simply use standard error-correcting codes to send all their information to the other side in a noise-robust way. Such a naïve approach would cause the conversation to be very long: the possibility of interacting is crucial for having efficient conversations. Consider, for instance, the preceding chess game. We can think of each player as having a fixed playing strategy that defines, for every position of the board, the next move that should be played. Compared to the (approximately) one hundred moves an average chess game takes [21], a description of a player's complete strategy may be extremely long as it needs to describe its move for all the (more than) 10^{40} different board positions [54].

A second naïve approach would be to employ error-correcting codes independently to each round of the conversation. Such an approach would result in poor performance—it could tolerate only a very small amount of noise, namely, the noise it takes to corrupt a *single* message. The ideal solution is one that tolerates a large amount of noise (e.g., a coding that works even if a constant fraction of the messages are corrupted), and yet does not increase the communication by too much (e.g., it multiplies the communication by at most a constant).

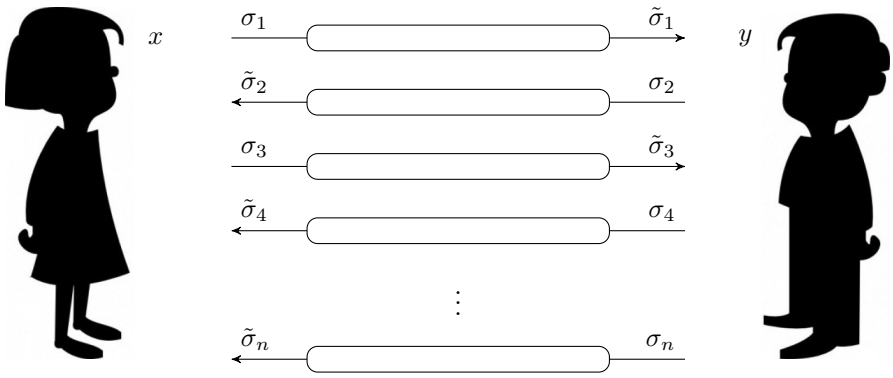

Figure 1.1: An alternating interactive protocol π of length $|\pi| = n$ rounds between Alice that holds the input x, and Bob that holds y, where the communication takes place over a noisy channel.

1.2 The setting

In the standard interactive communication setting [83], two parties (Alice and Bob) compute a function $f(x, y)$ by holding a conversation. Alice is given the input x, Bob is given y, and they aim to compute $f(x, y)$ by exchanging as few bits as possible. Kushilevitz and Nisan's book [60] gives an excellent description of the communication complexity of computing functions within the interactive setting. In the setting of *coding for interactive communication*, the channel that connects the parties may be noisy. The parties' goal is now to succeed, with high probability, in computing $f(x, y)$ despite the channel's noise, while sending as few bits as possible.

An interactive computation is performed via a *protocol* π, which is a pair of algorithms $\pi = (\pi_A, \pi_B)$ run by Alice and Bob, respectively. Each round, the protocol defines the next message to send, as a function of the party's input, the round number, and the symbols that party has received so far (the *transcript*). For example, in the first round Alice sends $\pi_A(x, 1, \emptyset) \in \Sigma$ and Bob sends $\pi_B(y, 1, \emptyset) \in \Sigma$, where Σ is the channel's alphabet. It is possible that only a single party speaks at each round—for example, Alice at odd rounds and Bob at even rounds. In this case we assume $\pi_A(x, i, \cdot) = \emptyset$ for even i's, and $\pi_B(x, i, \cdot) = \emptyset$ for odd i's.

After a fixed number n of rounds the protocol concludes and outputs a value. Alice's output is given by $\pi_A(x, n + 1, \mathsf{trans}_A)$, and Bob's, by $\pi_B(x, n + 1, \mathsf{trans}_B)$, where trans is the transcript seen by that party. Recall that due to the noisy channel, the parties may receive different symbols than the ones sent to them.

Let X, Y, Z be some finite sets. We say that π computes $f : X \times Y \to Z$ if for any inputs $(x, y) \in X \times Y$, both parties output $f(x, y)$. We define the length of the protocol to be its round complexity and denote it by $|\pi| = n$. At each round, the parties send a symbol out of the channels' alphabet Σ. The communication complexity of a protocol, $\mathsf{CC}(\pi)$, is the number of bits communicated by both parties; specifically, assuming one symbol is sent at each round, we have $\mathsf{CC}(\pi) = n \log |\Sigma|$.[1]

When evaluating a noise-resilient protocol π for some function f, it is convenient to compare it to the noiseless protocol π_0 of the same function. In this manuscript, we will care about *coding schemes* that, given a noiseless protocol π_0 construct a resilient version π that outputs a valid transcript of π_0 (and, thus, computes f). Our noiseless protocol π_0 is always defined over a binary

[1] Throughout this manuscript, all logarithms are taken to base two.

alphabet and takes $|\pi_0| = n_0$ rounds. Unless otherwise stated, Alice and Bob talk in π_0 in alternating rounds: Alice sends one bit in odd rounds and Bob sends one bit in even rounds, so that the total communication complexity of the noiseless protocol is $\mathsf{CC}(\pi_0) = n_0$. Note that any protocol can be converted into a binary alternating form at the cost of increasing the communication by at most $2\log|\Sigma|$. Hence, the preceding format of the noiseless protocol π_0 can be considered without loss of generality as its communication complexity differs by a factor of $2\log|\Sigma|$ (this difference does affect the generality when considering communication-optimized coding schemes). Due to technical reasons, we will sometimes need π_0 to be defined for rounds greater than n_0. In this case we can assume that after round n_0, π_0 sends zeros indefinitely.

Remark 1.1. In the following we will use the Landau notations to describe how a coding scheme π behaves with respect to the noiseless protocol π_0. In particular, we write $O(), \Omega()$, and so on, to denote the asymptotic behavior of quantities as $n_0 \to \infty$.

1.3 Parameters that we care about

We can evaluate the performance of an interactive coding scheme according to several parameters.

- **Maximal Noise Rate:** The maximal noise rate that the resilient protocol can tolerate. Usually, the noise rate $\varepsilon \in [0, 1]$ is measured as the fraction of corrupted symbols out of all the symbols that were communicated during the protocol. We will be mostly interested in coding schemes that tolerate a *constant fraction* of noise (i.e., when $\varepsilon = O(1)$). We also consider the case where the noise is stochastic (i.e., where each symbol is corrupted independently with probability ε; see §1.4).

- **Code Rate:** The *rate* of the coding scheme π with respect to the noiseless π_0, defined by

$$r = \frac{\mathsf{CC}(\pi_0)}{\mathsf{CC}(\pi)}.$$

  The rate indicates how much redundancy was added in order to make the computation noise resilient. We will be mostly interested in resilient schemes that have a *constant rate* (positive rate), in which the communication complexity of the resilient scheme is at most a constant times more than the complexity of the noiseless computation, $\mathsf{CC}(\pi) = O(\mathsf{CC}(\pi_0)) = O(n_0)$. If a scheme does not have a constant rate, that is, when $\lim_{n_0\to\infty} r = 0$, we say the coding scheme has a *vanishing* rate.

- **Success Probability** The probability that both parties output the correct value. The probability is over the randomness of the protocol (if randomized) and the noise (if randomized). We aim to obtain coding schemes that succeed with exponentially high probability in the length of the noiseless protocol, $1 - 2^{-\Omega(n_0)}$.

- **Efficiency:** The computational efficiency of the protocol. We aim for protocols for which the next symbol can be computed in at most polynomial time in n_0 (assuming a black-box access to π_0, as the noiseless protocol by itself may be inefficient).

1.4 Parameters that make a difference

When defining the setting, several variables come into play. Many times, these seemingly meaningless tweaks make a substantial difference in the capabilities of coding schemes.

- **The Channel.** We assume a channel Ch over alphabet Σ is a causal function $\mathsf{Ch} : \Sigma \to \Sigma$, where each instantiation of the function may (implicitly) depend on all previous channel instantiations. The channel is characterized by the following parameters:

  - *Alphabet size.* While the channel is always assumed to have a fixed-size alphabet (which is independent of the function we compute), the specific size of the alphabet may affect the noise resilience of the coding scheme. It is common that the alphabet in use is determined as a function of the noise resilience, and as the resilience approaches the limit, the alphabet size increases. The most difficult setting is thus when the alphabet is set to be of size 2, that is, a binary channel.

  - *Noise (type).* In the standard noisy channel, the noise may substitute an input symbol $\sigma \in \Sigma$ into any other symbol $\sigma' \in \Sigma$. A different type of noisy channel is the *erasure channel*, $\mathsf{Ch} : \Sigma \to \Sigma \cup \{\perp\}$, in which the input symbol is either delivered without any disturbance or turns into a special erasure mark $\perp \notin \Sigma$. In the more general *channel with insertions and deletions*, the channel $\mathsf{Ch} : (\Sigma \cup \emptyset) \to (\Sigma \cup \emptyset)$ is allowed to completely remove transmitted symbols (so that the receiver will not be aware a symbol was sent to it) or inject new symbols (so that a symbol arrives at the receiver without the sender sending it).

  - *Noise (power).* The power of the noise can be classified into three main categories:

    (i) *adversarial noise*, where the adversarial channel is considered to be all powerful, and the only restriction on the noise is the total amount of corruptions the channel is allowed to make. As mentioned before, the corruption budget is usually given as a fraction of the total communication. That is, an adversarial noise rate of ε means that at most εn symbols can be corrupted.
    (ii) *computationally efficient noise*, where the adversarial channel is considered to be computationally limited, in addition to being restricted to corrupting an ε-fraction of the transmissions.
    (iii) *random noise*, where each symbol is disturbed with some fixed probability, independently of previous transmissions, that is, a memoryless channel. The prominent example is the *binary symmetric channel* with flipping probability $\varepsilon < 1/2$, denoted BSC_ε, where each bit goes through undisturbed with probability $1 - \varepsilon$ or gets flipped with probability ε, independently per transmission. Note that random noise is a special type of a computationally bounded noise (yet, there is no limit on the fraction of corrupted transmissions).

  - *Feedback.* In the case of channels with feedback, we assume the sender instantly learns the symbol received at the other side via a separate *noiseless* feedback channel. The feedback channel is not counted towards the communication complexity nor the corruption budget.

- **Order of Speaking.** The order of speaking, both in the noiseless protocol π_0 and in the simulation π, may have a great effect on the properties of the coding scheme—specifically, its rate and noise resilience. We distinguish the case of *fixed order of speaking* in which the party that sends a symbol at the i-th round is predetermined and independent of the inputs of the protocol and the observed noise, and the case of *adaptive order of speaking*, where each party independently determines whether to send a symbol at the next round according to its input and received transcript.

- **Shared Randomness.** Whether or not the parties begin the computation with a random string unknown to the adversarial channel may have an effect either on the maximal obtainable rate of the coding scheme or on its noise resilience. In a sense, having a shared randomness has a certain effect of converting adversarial noise into a random one [64]. Practically, the parties can use shared randomness to better detect corruptions, reducing bit flips into erasure marks (with high probability).

- **Number of Parties.** The above interactive setting can be augmented to include the multiparty case, where m parties $\{p_i\}$ hold a private input $\{x_i\}$, respectively, and wish to compute some function $f(x_1, \ldots, x_m)$ while communicating over a noisy network. The network's topology has an important effect on the coding scheme's properties.

1.5 Organization

We begin in **Section 2** by discussing coding for interactive communication in the presence of adversarial noise. We discuss the maximal noise that can be tolerated and show a scheme with an optimal resilience. To that end we discuss two main techniques for coding (tree codes and rewind-if-error) that will be used throughout the manuscript. In **Section 3** we discuss (computationally) efficient constructions of coding schemes. We begin with the random noise setting and show several relaxations to tree codes that yield an efficient coding scheme. We then turn to the adversarial noise setting and show that optimal noise resilience can be achieved by an efficient coding scheme, using list-decoding techniques.

An advanced family of coding schemes adaptively change their structure (i.e., their length and the order in which the parties speak) according to the observed noise. In **Section 4** we discuss two models for adaptive protocols and show that a better noise resilience can be achieved in each of these more general models. The rate of coding schemes is discussed in **Section 5**. In particular, we show coding schemes in the random noise setting, whose rate approaches one as the noise probability approaches zero. We also discuss the maximal possible rate, that is, the capacity of interactive communication over memoryless noisy channels.

Section 6 explores other types of noisy channels. In particular, we survey coding schemes for channels that allow a noiseless feedback, erasure channels, channels with insertions and deletions, and quantum channels. In **Section 7** we extend the discussion to the multiparty case and discuss how to code distributed protocols performed over a noisy network, both in the random and the adversarial noise settings. When more than two parties perform a distributed computation, it is important to define whether messages pass in a synchronous or an asynchronous way. We survey coding schemes in both message-passing models and compare their properties. Applications of

coding for interactive communications, and related topics that build on the techniques of interactive coding are presented in **Section 8**.

Finally, in **Appendix B** we provide several tables that summarize the coding schemes discussed in this manuscript and compare their properties.

2

Noise Resilience

In this chapter we discuss the noise resilience of interactive coding schemes. We describe two simulation paradigms. The first, used, for example, in [76, 14, 19, 15, 17], relies on a special type of online coding known as a *tree code* [75, 76]. The second paradigm, *rewind-if-error* [74, 59, 51, 38, 27], is based on performing the original noiseless protocol until an error is detected and then rewinding to an earlier point, hopefully removing the incorrect simulation suffix.

2.1 The tree codes paradigm

Recall the chess game described in §1.1. When the noise level is large enough to corrupt even a single message ("`B2 to B4`" becoming "`D2 to D4`"), the entire simulation fails, since the parties' view becomes inconsistent. However, assume we have a machinery that allows the parties *eventually* to correctly decode any message that was sent to them. That is, as time goes by, earlier messages become more and more likely to be decoded correctly. In this case, earlier errors will eventually be detected, and subsequently the course of the simulation can be corrected.

Several properties are needed from such a machinery:

1. *Being online*: The encoding of each transmission can depend only on previous transmissions (but not on future ones).

2. *Distance:* The encodings of two different sequences of transmissions need to have a large distance (as a function of the sequence length), regardless of previous transmissions, location in the protocol, and so on.

The first property is critical for allowing the parties to use the code in an interactive setting: unlike the standard model, here the parties generate one transmission at a time, and need to communicate a transmission before generating the next one. The second property allows a party to eventually decode a transmission correctly, as long as the noise is low enough: the encodings of the correct sequence (sent by the other side) and the sequence the receiving side decodes are either the same, or they have a large distance from each other. Eventually, as time goes by, so does their distance, and if the noise is not too high, then the receiving side will be able to overcome it and correctly decode

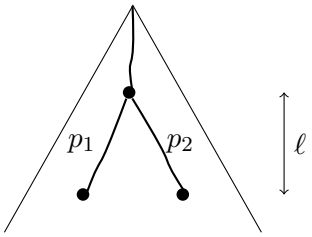

Figure 2.1: The labels of two diverging paths of the same length in a tree code satisfy $\Delta(label(p_1), label(p_2)) \geq \alpha\ell$.

a prefix of the sequence. At a given time, it may be that the noise at the suffix of the sequence is too high to decode the entire sequence correctly, but again, as time goes by, the same reasoning applies, and a longer correct prefix can be decoded.

Achieving the above properties is done via a *tree code*.

2.1.1 Tree codes: Definiton

Before describing the notion of tree codes, let us fix some notations. Let $\mathcal{T}$ be a rooted d-ary tree. Each node has d children ordered in a fixed way. Let $[d]$ denote the set $\{1, 2, \ldots, d\}$. We identify a node, with the rooted path leading to it, in an inductive manner. The root is identified with the empty path. The path $(e_1, \ldots, e_n)$, where $e_i \in [d]$, is identified with the e_n-th child of the node at the end of the rooted path $(e_1, \ldots, e_{n-1})$.

Definition 2.1 (tree code)**.** A d-ary tree code over an alphabet Σ with distance parameter α is an infinite rooted d-ary tree in which each edge e is marked with a label $w_e \in \Sigma$; for any two rooted paths $p_1 = (e_1, \ldots, e_n)$ and $p_2 = (e'_1, \ldots, e'_n)$ of the same length, that differ starting from the j-th level (i.e., $e_i = e'_i$ for all $i < j$, but $e_j \neq e'_j$), it holds that

$$\Delta\left(label(p_1), label(p_2)\right) \geq \alpha(n - j + 1).$$

Here, $label(p)$ denotes the concatenation of the labels along the path p, and $\Delta(x, y)$ is the Hamming distance between strings x and y of the same length, that is, the number of indices i for which $x_i \neq y_i$.

The definition suggests that the labels of any two paths diverging from the same node have a relative distance of at least α. That is, the Hamming distance of the labels of such paths is proportional to their length beyond the divergence point. See Figure 2.1.

In the following §2.1.2 we formally define tree codes, prove their existence and some of their properties. A reader that is mainly interested in an overview of coding scheme may consider skipping the detailed proofs of the next subsection and continue with §2.1.3 where we describe interactive coding schemes based on tree codes.

2.1.2 Properties of Tree Codes

In this section we extensively discuss the notion of tree codes: we show that tree codes (of unbounded depth) exist, prove certain properties of tree codes, and show equivalent definitions for tree codes.

Tree codes: definition via suffix distance

An equivalent and very useful definition for tree codes states that a tree code is one for which the labels of any two different paths have a large *suffix distance*.

Definition 2.2 (suffix distance)**.** For any two strings $x, y \in \Sigma^n$, the relative suffix distance is defined as

$$\Delta_{\mathsf{sfx}}(x, y) = \max_{i=1}^{n} \frac{\Delta(x_i \cdots x_n, y_i \cdots y_n)}{n - i + 1}.$$

We show that the above two tree code definitions are equivalent.

Lemma 2.1. A tree $\mathcal{T}$ is a tree code with distance α if and only if for any two different rooted paths p_1, p_2 of the same length in $\mathcal{T}$, $\Delta_{\mathsf{sfx}}(label(p_1), label(p_2)) \geq \alpha$.

Proof. First, we assume that $\mathcal{T}$ is a tree code and show that the suffix distance condition holds. Assume two paths that diverge at level $1 \leq j \leq n$; that is, $p_1 = (e_1, \ldots, e_{j-1}, e_j, \ldots, e_n)$ and $p_2 = (e_1, \ldots, e_{j-1}, e'_j, \ldots, e'_n)$ with $e_j \neq e'_j$. Let $label(p_1) = l_1^1 \cdots l_n^1$ and $label(p_2) = l_1^2 \cdots l_n^2$. Then,

$$\begin{aligned}
\Delta_{\mathsf{sfx}}(label(p_1), label(p_2)) &\geq \Delta_{\mathsf{sfx}}(l_j^1 \cdots l_n^1, l_j^2 \cdots l_n^2) \\
&\geq \frac{\Delta(l_j^1 \cdots l_n^1, l_j^2 \cdots l_n^2)}{n - j + 1} \\
&\geq \alpha,
\end{aligned}$$

where the last transition follows from the definition of $\mathcal{T}$ as a tree code with distance α.

As for the other direction, now assume that the suffix distance condition holds for any two different rooted paths in the tree. Let p_1, p_2 be rooted paths as before. We will show that $\Delta(label(p_1), label(p_2)) \geq \alpha(n - j + 1)$. For $i \geq 1$, define

$$\delta_i = \Delta(l_1^1 \cdots l_{j-1+i}^1, l_1^2 \cdots l_{j-1+i}^2) \quad \text{and} \quad s_i = \Delta_{\mathsf{sfx}}(l_1^1 \cdots l_{j-1+i}^1, l_1^2 \cdots l_{j-1+i}^2).$$

Additionally, define $\delta_0 = 0$.

We show, by induction on i, that $\delta_i \geq \alpha i$, which will prove the claim. Note that the claim trivially holds for $i = 0$. For $i = 1$, it is clear that $\delta_1 = s_1 \geq \alpha$. For $i > 1$, assume that the hypothesis $\delta_{i'} \geq \alpha i'$ holds for any $0 \leq i' < i$. Let $k \in [1, i]$ be the offset (beyond level $j - 1$) that maximizes s_i, that is, for which $\Delta(l_{j-1+k}^1 \cdots l_{j-1+i}^1, l_{j-1+k}^2 \cdots l_{j-1+i}^2) \geq \alpha(i - k + 1)$. We can use the induction hypothesis on levels $1, \ldots, j - 1 + (k - 1)$, i.e., with $i' = k - 1$. The induction hypothesis in this case implies that $\Delta(l_1^1 \cdots l_{j-1+(k-1)}^1, l_1^2 \cdots l_{j-1+(k-1)}^2) \geq \alpha(k - 1)$. It follows that

$$\begin{aligned}
\delta_i &= \Delta(l_1^1 \cdots l_{j-1+(k-1)}^1, l_1^2 \cdots l_{j-1+(k-1)}^2) + \Delta(l_{j-1+k}^1 \cdots l_{j-1+i}^1, l_{j-1+k}^2 \cdots l_{j-1+i}^2) \\
&\geq \alpha(k - 1) + \alpha(i - k + 1) \\
&\geq \alpha \cdot i.
\end{aligned}$$

□

Tree codes: existence

Schulman [76] used a simple probabilistic argument to show that for any distance parameter $\alpha < 1$, and any fixed degree of the tree, one can find an *infinite* tree code over an alphabet whose size depends on the degree of the tree and on α.

Theorem 2.2 (existence of tree codes [75, 76]). For any $\alpha < 1$ there exists a d-ary tree code of infinite depth and distance α over alphabet of size $|\Sigma| = (cd)^{1/(1-\alpha)}$, for some constant $c < 6$.

Proof. Assume a finite field $\mathbb{F}$ whose size we determine soon; $\mathbb{F}$ serves as the alphabet of the tree code. Let $g = g_1 g_2 g_3 \cdots$ be an infinite sequence with $g_i \in \mathbb{F}$. The string g is used to generate the labels of the tree in the following way: the edge at the end of the path $p = (e_1, e_2, \ldots, e_n)$ is labeled by the label w_p given by

$$w_p = \sum_{i=1}^{n} e_i g_{n-i+1}. \tag{2.1}$$

Recall that $e_i \in [d]$ is the edge going to the e_i-th child of the node at the end of the path $(e_1, \ldots, e_{i-1})$.

Next we pick g in a random way and show that there is positive probability of choosing g so that the tree code conditions are satisfied indefinitely. Assume we pick each g_i uniformly and independently from $\mathbb{F}$. Assume two different paths $p_1 = (e_1, \ldots, e_n)$ and $p_2 = (e'_1, \ldots, e'_n)$ of the same length that differ starting from the j-th level. Note that $label(p_1)$ and $label(p_2)$ are identical until the $(j-1)$-th index, and differ at the j-th index. After that point, these strings differ in their k-th index, $k \geq j$, if and only if

$$\sum_{i=j}^{k} e_i g_{k-i+1} \neq \sum_{i=j}^{k} e'_i g_{k-i+1},$$

or, alternatively, if

$$\sum_{i=j}^{k} (e_i - e'_i) g_{k-i+1} \neq 0. \tag{2.2}$$

Let X_k be the indicator that obtains the value 1 when the k-th label of p_1 differs from the k-th label of p_2. We know that $X_k = 0$ for all $k < j$. For $k \geq j$, recall that we pick each g_i uniformly from $\mathbb{F}$. Since $e_j \neq e'_j$, it is easy to see that $\mathbb{E}[X_k] = 1 - 1/|\mathbb{F}|$, since by fixing $g_1, \ldots, g_{k-j}$, there is exactly one value of g_{k-j+1} that makes the right-hand side of Equation 2.2 equal to 0. This also shows that X_i and $X_{i'}$ are independent for any $i \neq i'$ as long as both are at least j. Finally, observe that $X_j, \ldots, X_n$ depend only on $g_1, \ldots, g_{n-j+1}$, that is, only on the prefix of g whose length $n-j+1$ is the length of the divergent suffixes of p_1 and p_2 and on the difference of the labels of the suffixes, that is, on $(e_j - e'_j), \ldots, (e_n - e'_n)$ *but not on the location of these parts within the tree nor on the specific value of the labels.*

A specific g is called *bad* for some (p_1, p_2) of length n that differ starting at their j-th edge if $\sum_{i=j}^{n} X_i < \alpha(n-j+1)$. We can bound the probability for g to be bad for a specific (p_1, p_2) using Chernoff's bound by the following:

$$\Pr[g \text{ is bad for } (p_1, p_2)] \leq \Pr\left[\sum_{i=j}^{n} X_i \leq \alpha(n-j+1)\right] \leq e^{-(n-j+1)D(\alpha \,\|\, 1-1/|\mathbb{F}|)},$$

where $D(x\|y) \triangleq x \ln \frac{x}{y} + (1-x) \ln \frac{1-x}{1-y}$ and e is the base of the natural logarithm (not to be confused with e_i, which is the i-th edge).

Next, we union bound over all the different pairs of paths of depth n. As mentioned before, the probability depends on the difference of the labels of the divergent suffixes (of length $n-j+1$) of p_1 and p_2, and there are at most d^{n-j+1} different values the sequence $(e_j - e'_j), \ldots, (e_n - e'_n)$ can

take. Thus the probability that some generator g is bad for some (p_1, p_2) of any depth, is bounded by

$$\sum_{n-j=0}^{\infty} d^{n-j+1} \cdot e^{-(n-j+1)D(\alpha \,\|\, 1-1/|\mathbb{F}|)} \leq \sum_{n-j=0}^{\infty} e^{(n-j+1)\left(\ln d - \alpha \ln \frac{\alpha}{1-1/|\mathbb{F}|} - (1-\alpha)\ln \frac{1-\alpha}{1/|\mathbb{F}|}\right)}$$

which is strictly less than one, given that $|\mathbb{F}| > (ed)^{1/(1-\alpha)}$. Since the probability of g being bad is strictly less than one, there is a strictly positive probability for a good (infinite) generator g, which proves its existence. Since finite fields assume only sizes that are powers of primes, it is possible to find a finite field of size at most $(2ed)^{1/(1-\alpha)}$, which completes the proof. □

Although we know tree codes exist, we don't know any explicit construction for tree codes other than exhaustive search; in particular, no *efficient* construction is known to date. Braverman [14] showed, for any $\varepsilon > 0$, how to construct tree codes of depth n and distance up to approximately $1/10$ in time $2^{O(n^\varepsilon)}$. Moore and Schulman [66] gave an efficient construction of a code, that satisfies the tree code condition if a certain exponential sum assumption holds. Pudlák [71] drew a connection between certain types of linear tree codes and certain triangular matrices. Several relaxations of tree codes can be constructed efficiently [77, 70, 14, 42]; we discuss these relaxations in §3.1 and §3.2.

Tree codes: encoding and decoding

A tree code $\mathcal{T}$ is used to encode messages, symbol by symbol, in an online fashion. A d-ary tree code $\mathcal{T}$ implies an encoding

$$\mathsf{TCenc}_{\mathcal{T}}(m) = w_1 w_2 \cdots w_n$$

where w_i is the i-th label along the rooted path defined by the message $m = m_1 \cdots m_n$, where $m_i \in [d]$. Note that the tree code encoding is online (i.e., the code is a prefix code): the encoding of $m_1 \cdots m_i$ depends only on $m_1 \cdots m_i$ and is independent of any m_j with $j > i$. Indeed, for any message m and symbol $\sigma \in [d]$, it holds that

$$\mathsf{TCenc}_{\mathcal{T}}(m\sigma) = \mathsf{TCenc}_{\mathcal{T}}(m) \circ w_\sigma,$$

where w_σ is the label at the end of the rooted path $m\sigma$. This "online" property allows the usage of tree codes in interactive communication: recall that during an interactive protocol, only at round i does the party get the i-th symbol to communicate. By that time it has already communicated the symbols $\mathsf{TCenc}(m_1 \cdots m_{i-1})$, and in order to extend its message to $m_1 \cdots m_i$, the party needs to communicate only a single additional symbol. When the tree code in use is clear from the context, we will omit the subscript $\mathcal{T}$.

Decoding a received codeword $r \in \Sigma^n$ using the tree code $\mathcal{T}$ amounts to returning the message $m \in [d]^n$ whose encoding minimizes the Hamming distance to the received string,

$$\mathsf{TCdec}_{\mathcal{T}}(r) = \operatorname*{argmin}_{m \in [d]^n} \Delta(\mathsf{TCenc}_{\mathcal{T}}(m), r).$$

We now show that as long as the suffix distance between a received and sent words is below half the distance of the tree, then the tree succeeds in decoding *the entire sent word*.

Lemma 2.3. Assume $\mathcal{T}$ is a tree code with distance α over alphabet Σ. For any $r \in \Sigma^n$, such that

$$\Delta_{\mathsf{sfx}}(r, \mathsf{TCenc}_{\mathcal{T}}(m)) < \frac{\alpha}{2},$$

it holds that $\mathsf{TCdec}_{\mathcal{T}}(r) = m$.

Proof. Assume toward contradiction that $\mathsf{TCdec}_{\mathcal{T}}(r) = m'$ such that m' differs from m starting at the j-th index. Let $w = \mathsf{TCenc}_{\mathcal{T}}(m)$ and $w' = \mathsf{TCenc}_{\mathcal{T}}(m')$. By the definition of tree codes, we know that

$$\Delta(w_j \cdots w_n, w'_j \cdots w'_n) \geq \alpha(n - j + 1). \tag{2.3}$$

On the other hand, the lemma guarantees us that $\Delta_{\mathsf{sfx}}(r, \mathsf{TCenc}_{\mathcal{T}}(m)) < \alpha/2$; thus $\Delta(w_j \cdots w_n, r_j \cdots r_n) < \alpha(n - j + 1)/2$. Yet $\mathsf{TCdec}_{\mathcal{T}}(r) = m'$; thus w' minimizes the hamming distance to r, and since w and w' are the same up to the j-th index, we have that

$$\Delta(w'_j \cdots w'_n, r_j \cdots r_n) \leq \Delta(w_j \cdots w_n, r_j \cdots r_n) < \frac{\alpha(n - j + 1)}{2}.$$

This, along with a triangle inequality, implies that $\Delta(w_j \cdots w_n, w'_j \cdots w'_n) < \alpha(n - j + 1)$, contradicting our assumption in Equation 2.3. □

A simple converse follows immediately from the definition of tree codes,

Lemma 2.4. Assume $\mathcal{T}$ is a d-ary tree code with distance α, over alphabet Σ. Let $m \in [d]^n$ be some message, and let $s = \mathsf{TCenc}_{\mathcal{T}}(m)$ be its encoding using the tree code $\mathcal{T}$. For any $r \in \Sigma^n$ such that $\mathsf{TCdec}_{\mathcal{T}}(r)$ differs from m starting at the j-th index, it holds that

$$\Delta(s_j \cdots s_n, r_j \cdots r_n) \geq \frac{\alpha(n - j + 1)}{2}.$$

2.1.3 Interactive coding via tree codes

We now describe a coding scheme by Braverman and Rao [18] that has an optimal noise resilience while maintaining a constant rate. Specifically, for any $\varepsilon > 0$, it correctly computes the transcript of π_0 as long as the fraction of errors is at most $1/4 - \varepsilon$. The scheme builds on a previous scheme by Schulman [75, 76] that initially used tree codes (along with a rewind-if-error technique) in order to perform interactive communication over noisy channels.

We assume π_0 is an alternating binary protocol (recall that any protocol can be turned into an alternating binary protocol by increasing its communication by a factor of at most 2). We can visualize the noiseless protocol as the *protocol tree* $\mathcal{T}_0$—a binary tree in which Alice owns the odd levels and Bob, the even levels. For any input x, the protocol π_0 defines a set E_x of edges that correspond to possible replies by Alice (and, respectively, E_y for edges owned by Bob, for the input y). For any input (x, y), the edges of $E_x \cup E_y$ define a unique rooted path P_0, which corresponds to the correct transcript of $\pi_0(x, y)$. See Figure 2.2. The goal of the coding scheme is to reconstruct P_0 in the noisy setting. The task of finding P_0 when Alice is given E_x and Bob is given E_y is sometimes called the *pointer chasing* problem or the *pointer jumping* game. We emphasize that a protocol that solves the pointer chasing game for any input assuming a noisy communication is, in fact, a coding scheme that works for any π_0; that is, the pointer chasing problem is a complete problem for interactive coding.

The simulation goes as follows.[1] At each round, both parties send one edge from E_x or E_y, that corresponds to the next edge of P_0 that they own and haven't communicated in previous rounds.

[1] Here and throughout this manuscript, we always describe a coding scheme from Alice's point of view. The protocol for Bob is usually symmetric.

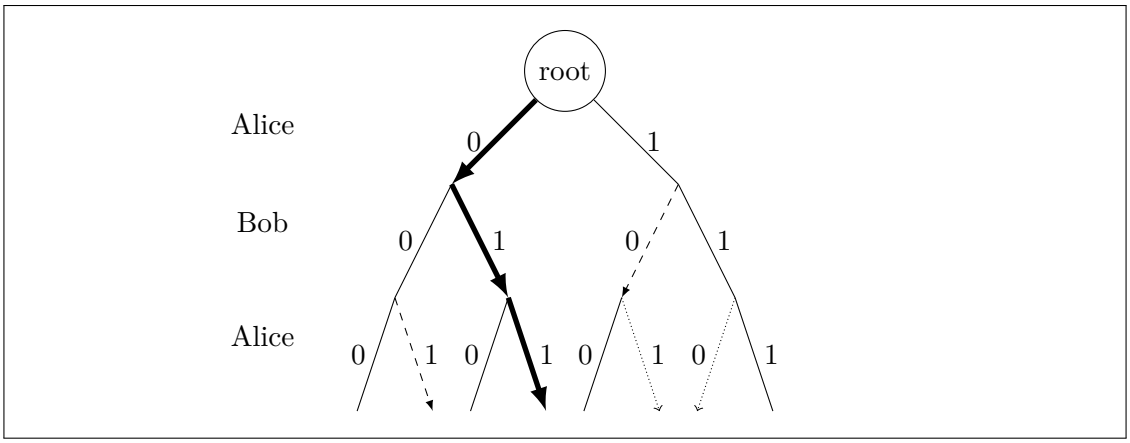

Figure 2.2: A tree $\mathcal{T}_0$ illustrating the correct path P_0 (bold edges) taken by Alice and Bob for the input (x, y). Edges with arrows in odd levels belong to E_x, and edges with arrows at even levels belong to E_y.

However, the noise might prevent them from learning the edges sent by the other side, so they do not learn the correct prefix of P_0, and send some edge that lies on a different path.

More formally, at each round, Alice maintains a set S_A of all the edges she has sent to Bob so far. At each round Alice decodes all the incoming messages from Bob and generates a (partial) guess $\tilde{S}_B$ of the set of edges S_B Bob has sent so far. This guessed set may include (some of the) edges of S_B, and possibly other edges added by the noise. Whenever $E_x \cup \tilde{S}_B$ contains a unique rooted path, Alice communicates the (unique) edge of E_x on that path that wasn't already communicated in previous rounds. If there is no unique path or Alice has no new edge to send, she sends some default symbol $\perp$.

Alice and Bob do not send edges in the clear, but rather they encode them using a tree code. Such an encoding allows them to (eventually) reconstruct the correct edges sent by the other side, despite the noise. The key point here is that the parties never send incorrect information: even if Alice guesses a wrong $\tilde{S}_B$ (due to noise), she always sends an edge that belongs to E_x. It can only be that the edge Alice sends is not on P_0 and thus it may be useless for Bob and for the simulation. Yet, as time goes by, the tree code guarantees that Bob correctly decodes a longer and longer prefix of the sequence of edges sent by Alice, and since this sequence always contains edges in E_x, then Bob eventually will be able to decode the correct prefix of P_0 and send his edge that continues that prefix.

Note that the message of Alice is of length $O(\log n)$ bits. Indeed, to extend a path, Alice needs to indicate only how to proceed from the deepest edge she owns in that path, that is, from the edge in S_A that appears at most two levels above the new edge to send. Since that edge was already communicated, say in the i-th transmission, Alice can communicate the new edge by sending the pointer i and a path of length at most 2 descending from her i-th transmitted edge. Therefore, to support n rounds, we obtain the message space $M = [n] \times (\{0,1\} \cup \{0,1\}^2)$ along with the symbol $\perp$. The length of each message is thus logarithmic in the length of the protocol, but with a simple encoding it can become constant. The key idea is that most of the messages point to a very recent edge, and only a few point to edges that were sent far away in the history. Then, if we encode each message using a relative pointer (i.e., the message 1 denotes the previously sent edge, instead of the first edge; the message i denotes the edge sent i transmissions ago rather than the i-th edge,

etc.), then the expected value of the pointer is a constant. If we encode each message $m \in M$ as a delimited binary string (e.g., over the alphabet $\{0, 1, \mathsf{sep}\}$), then most of the messages will be very short, and only a few will be of logarithmic length. Over the entire protocol, the *amortized* length of these strings is a constant. With the above encoding, each symbol comes from a ternary alphabet $\{0, 1, \mathsf{sep}\}$, and we can use a ternary tree code to encode each symbol.

Since converting the large-alphabet (with $O(\log n)$ bit symbols) into a constant size one costs only a constant in the rate of the scheme, we continue the analysis using the large alphabet, and show that for any noise rate $1/4 - \varepsilon$, running the simulation for $n = O_\varepsilon(n_0)$ rounds guarantees the successful simulation of the noiseless π_0. This yields the following theorem.

Theorem 2.5 ([19])**.** For any $\varepsilon > 0$, there exists an interactive coding scheme that simulates any protocol π_0 in $O_\varepsilon(n_0)$ rounds, uses a constant-size alphabet, and is resilient to a fraction $1/4 - \varepsilon$ corrupted symbols.

The simulation protocol (using polynomial-size alphabet) is sketched in Algorithm 1. The description is given for Alice; Bob's protocol is symmetric.

Algorithm 1 The Braverman-Rao simulation [18]

Input: a protocol π_0 of length $n_0 = |\pi_0|$ and an input x; a noise-resilience parameter $1/4 - \varepsilon$.
Let $\mathcal{T}$ be a d-ary tree code with distance $\alpha = 1 - \varepsilon$, depth $n = n_0/\varepsilon$, and $d = O(n)$.
Let E_x be the set of edges in $\mathcal{T}_0$ that correspond to Alice's possible replies, assuming she holds the input x.

$recv \leftarrow \emptyset$; $S_A \leftarrow \emptyset$
for $i = 1$ to $n = n_0/\varepsilon$ **do**
  $\tilde{S}_B \leftarrow \mathsf{TCdec}_{\mathcal{T}}(recv)$ ▷ Interpret each symbol as an edge ▷ (discard invalid/inconsistent edges)
  if $E_x \cup \tilde{S}_B$ has a unique rooted path P **then**
    $e_i \leftarrow$ the edge of $E_x \cap P$ with the lowest depth that is not in S_A
    $S_A \leftarrow e_i$
  else
    $e_i \leftarrow \perp$
  send the last symbol of $\mathsf{TCenc}_{\mathcal{T}}(e_1 \cdots e_i)$
  receive a symbol r from Bob
  $recv \leftarrow recv \circ r$
output the unique rooted path (of length n_0) defined by $E_x \cup \tilde{S}_B$

Before analyzing Algorithm 1 we give the following technical lemma showing that if the noise is bounded, there are many rounds in which the suffix distance between a sent and received codeword is small. By Lemma 2.3 this implies that at each such round, the entire word that was sent up to that round is correctly decoded. Formally, we have the following lemma.

Lemma 2.6. For any $r, s \in \Sigma^n$, if $\Delta(r, s) = \beta n$, then there exists a set of indices $I \subseteq [n]$ of size $|I| \geq (1 - \beta/\alpha)n$ such that for any $i \in I$,

$$\Delta_{\mathsf{sfx}}(r_1 \cdots r_i, s_1 \cdots s_i) < \alpha.$$

Proof. Consider the following algorithm that constructs I. We begin with $I = \emptyset$ and set $i = n$. If $\Delta_{\mathsf{sfx}}(r_1 \cdots r_i, s_1 \cdots s_i) < \alpha$, add i to I and set $i \leftarrow i - 1$. Otherwise, find the maximal $j < i$ such that $\Delta_{\mathsf{sfx}}(r_1 \cdots r_j, s_1 \cdots s_j) < \alpha$, set $i \leftarrow j$. Recurse on inputs $r_1 \cdots r_i$ and $s_1 \cdots s_i$.

Note that for each iteration in which we find j and disregard the indices $[j+1, i]$, it holds that $\Delta(r_{j+1} \cdots r_i, s_{j+1} \cdots s_i) \geq \alpha(i-j)$. Yet the maximal distance is bounded; thus summing over all the $n - |I|$ indices we remove, their accumulated distance cannot surpass βn. Therefore,

$$\alpha(n - |I|) \leq \beta n,$$

which completes the proof. □

Proof. (Theorem 2.5) Algorithm 1 succeeds due the following reasoning. Let s_A, s_B be the n-symbols Alice and Bob, respectively, send during the protocol and let r_A and r_B be the n symbols they receive. Also, let us denote the fraction of noise Alice sees by $\beta_A = \frac{\Delta(s_B, r_A)}{n}$ and, similarly, denote the noise Bob sees by $\beta_B = \frac{\Delta(s_A, r_B)}{n}$.

Since the total noise is bounded by $1/4 - \varepsilon$, then

$$\frac{\Delta(s_A, r_B) + \Delta(s_B, r_A)}{2n} = \frac{\beta_A + \beta_B}{2} \leq \frac{1}{4} - \varepsilon.$$

Recall that the users use a tree code with distance $\alpha = 1 - \varepsilon$. Using Lemma 2.6, we know that for Alice there will be $|I_A| \geq (1 - \frac{2\beta_A}{1-\varepsilon})n$ iterations i in which $\Delta_{\mathsf{sfx}}(r_A[1..i], s_B[1..i]) < \alpha/2$, and for Bob, $|I_B| \geq (1 - \frac{2\beta_B}{1-\varepsilon})n$ iterations i's in which $\Delta_{\mathsf{sfx}}(r_B[1..i], s_A[1..i]) < \alpha/2$. In each of these rounds, Lemma 2.3 tells us that the parties correctly decode the entire message sent to him by that round. It holds that $|I_A \cap I_B| \geq \varepsilon n$:

$$\begin{aligned}
|I_A| + |I_B| &= \left(2 - \frac{2\beta_A + 2\beta_B}{1-\varepsilon}\right) n \\
&\geq 2n - \frac{1-4\varepsilon}{1-\varepsilon} n \\
&\geq 2n - (1-3\varepsilon)n \\
&\geq n(1+3\varepsilon).
\end{aligned}$$

Now, note that as long as Alice correctly decodes a message that Bob sent, she can extend the (correct!) joint path by another edge. Then, the next time Bob decodes the correct message from Alice, he will be able to see that (correct!) path and extend it by another edge. Since $|I_A \cap I_B| > \varepsilon n = n_0$, at each such iteration the correct path is extended by at least one edge; thus we are guaranteed that by round n, each of the parties has sent all its edges on the correct path.

The last part is to establish that the parties, at round n, decode long-enough prefixes of the sent message to allow them to decode the correct path. We stress that as long as a prefix that contains the entire path is decoded, all subsequent (possibly incorrect) edges will be discarded at line 3 of the algorithm.

Let i be the first round for which $s_A[1..i] \cup s_B[1..i]$ contains the entire correct path. As mentioned earlier, if at round n Alice decodes a word m such that $m[1..i] = s_B[1..i]$, then she outputs the correct value. Due to the properties of the tree code (Lemma 2.4), in order for Alice to decode a shorter prefix of s_B, it must hold that $\Delta(r_A[i..n], s_B[i..n]) \geq \alpha(n - i + 1)/2$.

However, we note that in every round where the parties did not advance their path, there must have been enough noise to prevent them from decoding the correct path so far. In other words, if i is large, there must have been many errors before round i. Specifically, we have the following.

Lemma 2.7. For i defined as above,

$$\Delta(s_B[1..(i-1)], r_A[1..(i-1)]) + \Delta(s_A[1..(i-1)], r_B[1..(i-1)]) > \frac{\alpha}{2}\left(i - 1 - \frac{n_0}{2}\right).$$

Proof. Denote

$$\beta'_A = \frac{\Delta(s_B[1..(i-1)], r_A[1..(i-1)])}{i-1}, \qquad \beta'_B = \frac{\Delta(s_A[1..(i-1)], r_B[1..(i-1)])}{i-1}$$

and assume without loss of generality that $\beta'_A \geq \beta'_B$. By the definition of the quantity i, it holds that up to round $i-1$ there were less than $n_0/2 = \varepsilon n/2$ rounds where Alice extends the correct path by one edge (due to our assumption, Alice is the party that stalls more). Lemma 2.6 now suggests that $(1 - \frac{2\beta'_A}{\alpha})(i-1) < \varepsilon n/2$, hence $\beta'_A > \frac{\alpha}{2}(1 - \frac{\varepsilon n}{2(i-1)})$. This implies that the number of corruptions that affect Alice's transmissions is at least $\beta'_A(i-1) > \frac{\alpha}{2}(i - 1 - \frac{\varepsilon n}{2})$. □

Therefore, the corruption budget left for rounds $[i, n]$ is bounded by

$$\begin{aligned} 2n\left(\frac{1}{4} - \varepsilon\right) - \frac{\alpha}{2}\left(i - 1 - \frac{\varepsilon n}{2}\right) &= n\left(\frac{1-\varepsilon}{2} - \frac{3}{2}\varepsilon\right) - \frac{\alpha}{2}(i-1) + \frac{\alpha\varepsilon n}{4} \\ &< \frac{\alpha n}{2} - \frac{\alpha}{2}(i-1) \\ &= \frac{\alpha(n-i+1)}{2}, \end{aligned}$$

and is too small to prevent Alice from correctly decoding a prefix of length i of s_B. Hence, Alice outputs the correct value. A similar argument holds for Bob. □

Remark 2.1. Algorithm 1 assumes a channel with polynomial alphabet size but that can be reduced to alphabet of size $O_\varepsilon(1)$, as mentioned before, by encoding each edge using a variable-length encoding over the alphabet $\{0, 1, \mathsf{sep}\}$. Each symbol is then encoded via a ternary tree code with distance α, and alphabet of size $3^{O(1/(1-\alpha))}$.

We can furthermore reduce the alphabet size and obtain an equivalent scheme that communicates bits (i.e., uses a binary alphabet). This is done by using a relaxed notion of tree code in which each label is a binary string. Such tree codes are shown to exist for any distance parameter $\alpha < 1/2$. The maximal noise resilience obtained by this approach is $1/8 - \varepsilon$ [19]. It is still an open question whether this resilience is optimal or not, as the best upper bound for the binary case [27] is $1/6$. See Theorem 6.3.

2.2 The rewind-if-error paradigm

The other main paradigm for performing resilient interactive communication is the *rewind-if-error* paradigm. The main idea here is very simple: the parties run the protocol π_0 for k rounds, after which they transfer some information in order to verify whether or not errors have occurred in the computation. If there is no indication of errors, the parties continue to simulate the next part of π_0. If, on the other hand, the parties believe there were errors, they simply rewind π_0 by k' rounds, removing its possibly incorrect suffix, and repeat. The main issue of this paradigm is how to check whether errors have happened or not and how to (simultaneously) agree whether to rewind or not

Rewind-if-error coding paradigm:

1. Run π_0 for k steps
2. Check whether the other side agrees on the simulated transcript so far.
   (a) If the other side seems to be synchronized: repeat (next k steps of π_0).
   (b) If there is an apparent mismatch: rewind k' rounds, and repeat.

Figure 2.3: The rewind-if-error paradigm

or, alternatively, how to regain synchronization when one party decides to rewind while the other does not. The outline of this paradigm is described in Figure 2.3

In order to verify whether the parties agree on the simulated transcript, the most common way is to exchange hashes of the simulated transcript. Specifically, assume the family of Hash functions given by Theorem 2.8.

Theorem 2.8 ([67, 3])**.** For any $k \in \mathbb{N}$, there exists a family of hash functions $\mathcal{H}_k = \{h_s : \{0,1\}^{2^k} \to \{0,1\}^{q \cdot k}\}$, where $|s| = q \cdot k$ for some constant q, such that for any two distinct $x, y \in \{0,1\}^{2^k}$ it holds that

$$\Pr_{s \in_U \{0,1\}^{qk}}[h_s(x) = h_s(y)] \leq 2^{-k}.$$

Furthermore, $h_s \leftarrow \mathcal{H}_k$ can be sampled efficiently, and $h_s(\cdot)$ can be computed efficiently (in 2^k).

If Alice holds a value x and Bob holds y, they can check whether they hold the same value (with high probability) by exchanging hashes: Alice samples s and sends Bob $s, h_s(x)$. Bob then computes $h_s(y)$ and compares it to the value he received from Alice. If $x = y$, then it is always the case that $h(x) = h(y)$, so Bob learns they have the same value. However, if $x \neq y$, then Bob learns their values are different with probability $1 - 2^{-k}$. This consistency check will be used by the coding scheme to identify errors that may have happened and cause a rewind as necessary.

To illustrate the ideas of the rewind-if-error paradigm we describe a very simple coding scheme for a slightly nonstandard π_0. We assume that π_0 takes n_0 alternating rounds, in each of which a symbol of length $\log n_0$ bits is sent; similarly, the coding scheme uses a large alphabet (a similar scheme first appeared in [51]). We show that for any $\varepsilon > 0$, the coding scheme described in Algorithm 2 is resilient to noise rates of up to $1/8 - \varepsilon$ and has a constant rate (given the nonstandard definition of π_0). Despite the nonstandard model and the fact this scheme does not obtain the optimal noise resilience, it is instrumental in conveying the ideas of the rewind-if-error paradigm that will be used extensively later.

Theorem 2.9. Algorithm 2 has a constant rate.

Proof. We assume that the entire information that is sent during a single iteration is considered as a single symbol of the alphabet. This information contains one symbol of π_0, a seed s, the length of the transcript simulated so far, and a hash value. The entire simulation sends $2n$ symbols—2 symbols in each iteration, one in each direction. Each sumbol takes $O(\log n_0)$ bits. Recall that π_0 communicates n_0 symbols, where each takes $\log n_0$ bits. Thus the rate of the coding scheme is given

Algorithm 2 A simplified rewind-if-error coding scheme with polynomial-size alphabet

Input: a protocol π_0 of length n_0 defined over symbols of size $\log n_0$ bits and an input x; a noise-resilience parameter $1/8 - \varepsilon$.

Let $\mathcal{H} = \{h_s : \{0,1\}^{2n_0} \to \{0,1\}^{O(\log n_0)}\}$ be a hash function family given by Theorem 2.8. Note that the required seed size is $|s| = O(\log n_0)$.

$T \leftarrow \emptyset$
for $i = 1$ to $n = n_0/8\varepsilon$ **do**
    sample s and send $\sigma = (s, |T|, h_s(T))$
    receive $\tilde{\sigma} = (\tilde{s}, \tilde{l}, \tilde{h})$
    run π_0 for a single round given the past T (send/receive one symbol); Let T' be the symbol sent or received
    if $|T| = \tilde{l}$ and $h_{\tilde{s}}(T) = \tilde{h}$ **then**
        $T \leftarrow T \circ T'$
    else
        if $|T| \geq \tilde{l}$, delete 1 symbol from the suffix of T
output T (a prefix of length n_0)

by

$$\frac{n_0 \cdot \log n_0}{2n \cdot O(\log n_0)} = \frac{n_0}{2(n_0/8\varepsilon) \cdot O(1)} = O_\varepsilon(1). \qquad \square$$

Theorem 2.10. Algorithm 2 is resilient to a fraction $1/8 - \varepsilon$ of corruptions, with probability $1 - 2^{-\Omega(n \log n)}$.

Proof. Again, recall e assume the entire information that is sent during a single iteration is considered as a single symbol of the alphabet. This information contains one symbol of π_0, a seed s, the length of the transcript simulated so far, and a hash value. Note that each of the preceding quantities takes $O(\log n_0)$ bits to describe. Therefore, each iteration can be considered to be sending a single symbol of alphabet of size $2^{O(\log n_0)} = \text{poly}(n_0)$ by each party.

Consider the state at the end of iteration i, and consider the transcripts both parties hold, say, $T_A(i)$ and $T_B(i)$. Compared to T_0, the transcript of π_0, the partial transcripts the parties hold may be correct until some point and different from that point an on.

Formally, for any iteration $i \in [n]$, we define a potential function $\varphi(i)$: let $m(i)$ be the maximal point where both $T_A(i), T_B(i)$ equal T_0 (measured in symbols of size $\log n_0$ bits, as sent by π_0). Then,

$$\varphi(i) = 3m(i) - |T_A(i)| - |T_B(i)|,$$

that is, the length (in blocks) of the "correct" prefix minus the incorrect suffixes. Our goal is to show that, at the end of the computation, the potential is large enough. Specifically, if $\varphi(n) > n_0$, it means that the correct prefix of T_A and T_B at the end of the computation is at least of length n_0 symbols, which implies the correct output.

Before analyzing the potential, let us first define the bad event of a *hash collision*, defined as the case where $h(x) = h(y)$ while $x \neq y$. Particularly, a hash collision means that even though the parties hold different transcripts, their consistency test fails to indicate this discrepancy. However, we can choose our hash function family $\mathcal{H}$ to be such that the probability of hash collision is $2^{-O(\log n_0)} \ll 1/n^2$. Throughout the protocol the parties perform at most $2n$ hash comparisons;

thus a union bound guarantees that the expected number of hash collisions throughout the protocol is negligible, $\ll 1/n$, and we can assume with high probability that no hash collision occurs during the protocol.

We now analyze the behavior of the potential function throughout the protocol. Assume that no errors happen during some iteration and there is no hash collision during this iteration. Then, the potential increases by one, $\varphi(i) \geq \varphi(i-1)+1$: either both parties are synchronized and they simulate the next step of π_0 (thus $m(i)$ increases by 1, as well as $|T_A(i)| = |T_B(i)|$, and the potential increases by $3-2=1$) or they realize they are not synchronized, in this case at least one of the parties removes one incorrect symbol from the suffix of its transcript, while $m(i)$ does not change. The other party may either remove k bits or keep its recorded transcript unchanged, but it is never the case that the recorded transcript extends.

On the other hand, no matter what error happens or if a hash collision happens or not, the potential can decrease by at most 3, $\varphi(i) \geq \varphi(i-1)-3$: the largest decrease happens if m is reduced by 1 (which means that one of the parties removed a (correct) symbol from T), while the other transcript increases by 1 (incorrect) symbol.

Finally, when the fraction of noise is bounded by $1/8-\varepsilon$, there are at most $n_{bad} = (1/8-\varepsilon)\cdot 2n$ erroneous iterations in which the potential decreases[2], and at least $n_{good} = n - n_{bad}$ iterations in which the potential increases. Therefore, the potential at the end of the computation satisfies

$$\begin{aligned}
\varphi(n) &\geq n_{bad} \times (-3) + n_{good} \times (+1) \\
&\geq -3n\left(\frac{1}{4}-2\varepsilon\right) + n\left(1-\left(\frac{1}{4}-2\varepsilon\right)\right) \\
&\geq 8\varepsilon n \\
&\geq n_0.
\end{aligned}$$

Therefore, the correct prefix of T_A, T_B at the end of the computation is at least n_0, except with probability at most $1/n$. If we wish to reduce the failure probability even further, we can set $n = n_0/\varepsilon$. In this case, even up to εn hash collisions happen throughout the protocol, it still holds that $\varphi(n) > n_0$. On the other hand, the probability of having so many hash collisions is clearly bounded by $2^{-\Omega(n \log n)}$. □

A resilience of $1/8-\varepsilon$ holds for polynomial alphabet but does not hold anymore for smaller alphabets. If we count the noise as the fraction of bits that were exchanged, then the scheme in Algorithm 2 is resilient to a fraction of noise of $O(1/\log n)$. It is possible, however, to get better resilience (i.e., resilience against some constant fraction of noise) [51] by employing a more sophisticated rewinding mechanism in which parties can revert to specific agreed "meeting points" [74, 51]. We discuss this mechanism in §5.2.

The obvious advantage of these coding scheme is that they do not relay on tree codes, and are more intuitive. They also serve a crucial role in obtaining coding scheme with a rate that approaches one; see §5.

[2]Basically, we lose a factor of 2 in the resilience since both parties speak at each iteration. However, this is necessary in order to ensure they are synchronized.

2.3 Upper bounds on the maximal noise

We conclude this chapter by showing that the coding scheme of Algorithm 1 actually achieves the maximal resilience possible for *nonadaptive* coding schemes.[3] In nonadaptive protocols the structure of the protocol is well-defined, regardless of the observed noise. Specifically,

Definition 2.3 (nonadaptive protocols)**.** A nonadaptive protocol is one in which for any input, and any noise pattern, there is a consensus between Alice and Bob about the next party to speak.

Braverman and Rao [19] prove that nonadaptive protocols must take the same number of rounds on every input, and that the party that speaks at every round is determined as a function of the round number alone (namely, is independent of the input and noise).

Lemma 2.11 ([19])**.** A nonadaptive protocol π satisfies the following: (1) for all inputs, the protocol runs for n rounds; (2) at any round, it is predetermined which party speaks (independent of input and noise)

Therefore, protocols in which the parties exchange symbols every round (or in alternating rounds), are trivially nonadaptive.

Remark 2.2. The above lemma suggests that nonadaptive protocols always have a fixed order of speaking, i.e., that the set of nonadaptive protocols and the set of *fixed-order* protocols is the same one. However this is not the case when other types of noisy channels are considered. Specifically, in §6 we discuses nonadaptive protocols whose order of speaking is not fixed.

The maximal resilience of nonadaptive protocols is $1/4$. Formally, we have the following.

Theorem 2.12 ([19])**.** No nonadaptive interactive protocol for the identity function $f(x, y) = (x, y)$ is resilient to noise rate of $1/4$ with probability greater than $1/2$ (in the plain model).

Proof. Assume a protocol π for the identity function, that communicates n symbols on every inputs (which must be the case for a nonadaptive protocol). Consider instance of the protocol on input $(0, 0)$ and assume without loss of generality that Alice is the party that sends $n_A \leq n/2$ symbols. Since the protocol is nonadaptive, the same holds also any instance of $\pi(1, 0)$.

Consider the following two attacks: (1) an instance of $\pi(0, 0)$ in which the channel changes Alice's first $n_A/2$ symbols to be exactly what Alice sends at the same rounds in $\pi(1, 0)$; (2) an instance of $\pi(1, 0)$ in which the channel changes Alice's last $n_A/2$ symbols to be exactly what Alice sends at the same rounds in the first instance.

It is clear that Bob's view in both instances (1) and (2) is the same; therefore, Bob gives the correct output with probability at most $1/2$. The total noise in both instances is $n_A/2 \leq n/4$. □

2.3.1 Circumventing the $1/4$ upper bound

It is possible to circumvent the above impossibility and achieve coding schemes with noise resilience above $1/4$ by either relaxing the noise model or the conditions of the impossibility bound of Theorem 2.12. Here, we explore one such relaxation and state several other possible relaxations that will be discussed later in this manuscript.

[3] In previous work [18, 19] nonadaptive protocols were also called *robust* or *oblivious*.

A crucial point in the proof of Theorem 2.12, is the ability to identify *in advance* the party that speaks less and attack that specific party. However if the party that "speaks less" changes as a function of the observed noise (i.e., when the protocol is adaptive), then the proof no longer holds, and indeed better resilience is achievable. We discuss the case of adaptive protocols in §4.

Another possible relaxation is to allow one of the parties (but not both!) to learn about an event of an error. This naturally happens in different noise models such as the erasure channels, or channels with noiseless feedback. See §6 for the maximal noise resilience in these settings.

Braverman and Efremenko [15] considered the case where there is a one limit for the fraction of noise on the channel from Alice to Bob and a different limit for the fraction of noise on the channel in the other direction, from Bob to Alice. Let the fraction of noise in one direction be α and in the other direction β. Then Algorithm 1 works as long as $\alpha + \beta < 1/2$. Braverman and Efremenko analyzed the exact region of noise that is feasible. That is, they show codings schemes for any point (α, β) within this region, and prove that no coding scheme succeeds if the noise (α, β) is outside that region. The feasible region nontrivially extends the bound $\alpha+\beta < 1/2$. In particular, $(1/3, 1/3)$ is feasible. The coding schemes of [15] use list-decoding techniques (see §3.4.1) and also let the parties speak a different number of rounds as a function of the specific noise (α, β).

Allowing the parties to preshare a random string that is unknown to the adversarial channel also allows lifting the maximal noise and yields a coding scheme with noise resilience of $1/2 - \varepsilon$, shown by Franklin, Gelles, Ostrovsky, and Schulman [34, 35]. [A similar resilience is achievable in the quantum world (where parties preshare entangled states) [13]; see §6.4.] The parties use the shared randomness in order to detect noise: each transmission is encoded via a random code whose randomness is taken from the shared string. Such a code serves as a message authentication code [47], ensuring that any corruption is detected with high probability. Specifically, consider the following random code known as the *blueberry code*.

Definition 2.4 (blueberry code [35])**.** Given alphabets Π, Σ, where $\Pi \subseteq \Sigma$, and n independent random permutations $B_i : \Sigma \to \Sigma$, the blueberry code $B : \Pi^n \to \Sigma^n$ is the mapping

$$B(x_1 \cdots x_n) = B_1(x_1) \cdots B_n(x_n).$$

At the i-th transmission, only a random subset of Σ whose size is $|\Pi|$ is being used by the code, namely, $S_i = \{B_i(x) \mid x \in \Pi\}$. The subset S_i is uniformly chosen from Σ and is unknown to the channel. Then, with probability $1 - \frac{|\Pi|-1}{|\Sigma|-1}$, the noise will alter the transmission into a symbol outside S_i and will be detected by the parties. In this case, the receiver knows that the received symbol is corrupted, so it marks the transmission by a special erasure mark, $\perp$. Since erasures are "twice as easy" to handle than errors, this effectively doubles the noise resilience (see also §6.2). Combining this approach with Algorithm 1 and carefully analyzing the effect erasures have on Algorithm 1 gives the following.

Theorem 2.13 ([35])**.** For any $\varepsilon > 0$, there exists a coding scheme that simulates any protocol π_0 in $O_\varepsilon(n_0)$ rounds, uses a constant-size alphabet and is resilient to a fraction $1/2 - \varepsilon$ of corrupted symbols, assuming the parties share a random string of length $O(n_0)$ unknown to the adversarial channel.

It is easy to see that $1/2$ is an upper bound on the noise resilience of nonadaptive coding schemes, regardless of any computational or other setting assumption.

Theorem 2.14 ([35])**.** No nonadaptive interactive protocol for the identity function $f(x, y) = (x, y)$ is resilient to a noise rate of $1/2$ with probability greater than $1/2$, even given any other setting assumption.

Proof. Let π be a protocol for the identity function. Since the protocol is nonadaptive, the length of the protocol is fixed. Furthermore, the order in which the parties speak is fixed and pre-determined (Theorem 6.4), which also determines the total number of symbols n communicated in the protocol. It follows that there exists a party (*without loss of generality*, Alice) that speaks at most $n/2$ of the symbols. The channel can completely corrupt every symbol sent by Alice. This incurs a noise rate of at most $1/2$, and prevents Bob from learning Alice's input. □

Open Questions for Section 2

1. What is the maximal noise resilience of *binary* protocols?
2. Find a binary coding scheme with a constant rate that achieves the maximal resilience.
3. Is it possible to obtain the maximal resilience in a coding scheme that has a constant rate and is solely based on the rewind-if-error paradigm?
   If not, what is the maximal noise resilience obtainable with a constant rate, and how do the resilience and the rate trade off?

3

The Hunt for Efficient Constructions

When using tree codes, two factors prevent coding schemes from being efficient. The first is that there is no known efficient construction of a tree code. However the second reason is the lack of efficient *decoding* procedures. That is, even if one has an efficient way to construct tree codes (i.e., an efficient algorithm that generates the label for any given path, TCenc), it is not clear how one can decode a given received word, that is, how to efficiently compute TCdec. For this reason, the running time of Algorithm 1 is exponential in the length of the simulation, in the worst case.

Luckily, when the noise is random, then the decoding issue no longer prevents us from achieving efficient coding schemes. Schulman [76] shows that in the random noise setting, one can efficiently compute TCdec in a greedy way, so that the expected decoding time is polynomial in the length of the word to be decoded. Still, the first issue prevails, namely, the inefficient construction of tree codes prevents us from obtaining a fully efficient scheme.

In this section we discuss efficient constructions of interactive coding schemes. First, we show that if we are willing to compromise some of the parameters (e.g., the rate or the failure probability) then efficient schemes can be found. Then, we discuss the case of random noise and provide efficient constructions of tree code–based schemes. Finally, we move to the realm of adversarial noise and describe a scheme that achieves it all: it is efficient, is resilient to the maximal possible noise, and succeeds with overwhelming probability.

3.1 Efficient schemes for random noise with reduced parameters

3.1.1 Coding schemes with a vanishing rate

Efficient coding schemes are quite easy to devise if one is willing to compromise the rate of the coding scheme and allow a vanishing rate. Ghaffari, Haeupler, and Sudan [46] note that a simplified version of Algorithm 1 is efficient and is resilient to the optimal fraction of noise, $(1/4 - \varepsilon)$. In this simplified version, each symbol of the alphabet is large enough to contain the entire set of edges S_A sent so far (rather than only a single edge, as done in Algorithm 1). Each such set contains at most

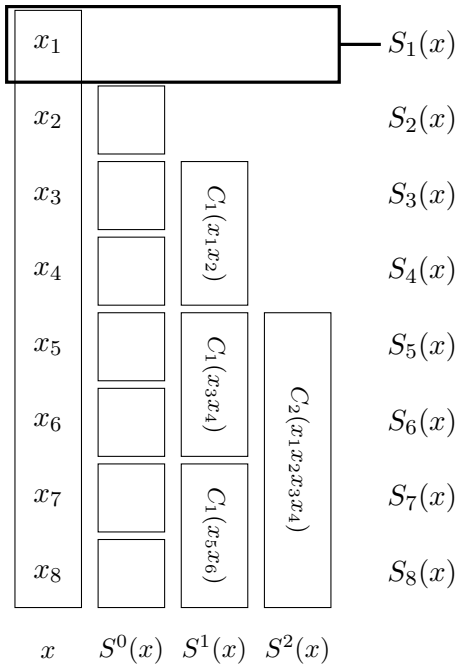

Figure 3.1: A simplified illustration of the tree code encoding $S(x) = \mathsf{TCenc}(x)$ of a string x of length 8. Each symbol $S_i(x)$ of the encoding consists of four symbols: the input x itself and $\log |x|$ symbols out of codewords of increasing lengths $1, 2, 4, ...$, encoding prefixes of x. [The shifted codewords $\tilde{S}^i(x)$ are omitted here.]

n edges (out of at most 2^{n_0} possible edges in $\mathcal{T}_0$), and takes $O(n_0)$ bits to describe. The obtained rate is then $O_\varepsilon(1/n_0)$, yet no tree codes are needed, and the scheme is computationally efficient.

In an unpublished note [77], Schulman describes a way to efficiently construct a tree code of depth n over an alphabet of polynomial size in n. Plugging such a tree code in Algorithm 1 (assuming a high-enough distance parameter α) gives, for any $\varepsilon > 0$, an efficient coding scheme with resilience $1/4 - \varepsilon$ and rate $O_\varepsilon(1/\log n_0)$. Based on ideas in [77, 14, 39], we now describe a simplified construction of a tree code of depth n and distance $\alpha < 1$ over an alphabet with polynomial size.

Theorem 3.1. For any $\alpha < 1$ there exists an efficient construction of a d-ary tree code of depth n over an alphabet of size polynomial in n.

Proof. The high-level idea is to construct $O(\log n)$ parallel encoding "layers", where each layer is an encoding of parts of the inputs via simple error-correction code. The block size doubles in each layer, that is, the first layer will be the input itself; the second layer separately encodes every two symbols of the input; the third layer encodes four symbols, etc. Since the encoding must be online, we can encode input symbols only after the time they arrive. Therefore, the encoding of x_1, x_2 (in the second layer) can be used only after x_2 is known. The first block in each layer is thus set to a constant. See Figure 3.1 for a visualization.

We now formally define the encoding and later prove it satisfies the distance requirements. For any $i < \log(n/2)$, assume an efficient error-correcting code $C_i : [d]^{2^i} \to \Sigma^{2^i}$ with relative distance at least δ; it is possible to find such a code (e.g., a Reed-Solomon code) with $|\Sigma| = O(d \cdot \frac{1}{1-\delta})$.

Let $0 \in \Sigma$ be some arbitrary symbol. For any $x \in [d]^n$ and any $i \leq \log(n/2)$, define the string $S^i(x) \in \Sigma^n$ as follows:

$$\begin{aligned}
S^0(x) &= 0 \circ x_1 \circ \cdots \circ x_{n-1} \\
S^1(x) &= 0^2 \circ C_1(x_1x_2) \circ C_1(x_3x_4) \circ \cdots \circ C_1(x_{n-3}x_{n-2}) \\
S^2(x) &= 0^4 \circ C_2(x_1x_2x_3x_4) \circ C_2(x_5x_6x_7x_8) \circ \cdots \circ C_2(x_{n-7}x_{n-6}x_{n-5}x_{n-4}) \\
&\vdots \\
S^i(x) &= 0^{2^i} \circ C_i(x_1 \cdots x_{2^i}) \circ \cdots \circ C_i(x_{n-2\cdot 2^i+1} \cdots x_{n-2^i}) \\
&\vdots \\
S^{\log(n/2)}(x) &= 0^{n/2} \circ C_{n/2}(x_1 \cdots x_{n/2}).
\end{aligned}$$

Additionally, for any $i \geq$ let $\tilde{S}^i$ be the string obtained by shifting S^i by 2^i indices, padding zeros at the beginning and trimming the suffix as needed, that is,

$$\tilde{S}^i(x) = 0^{2^i} \circ 0^{2^i} \circ C_i(x_1 \cdots x_{2^i}) \circ \cdots \circ C_i(x_{n-3\cdot 2^i+1} \cdots x_{n-2\cdot 2^i}).$$

Finally, let $S \in \Sigma^{1+2\log n}$ be the string $S = (S_1, S_2, \ldots, S_n)$ whose j-th index is the tuple

$$S_j = (x_j, S_j^0, S_j^1, \ldots, S_j^{\log(n/2)}, \tilde{S}_j^0, \ldots, \tilde{S}_j^{\log(n/2)}),$$

where S_j^i and $\tilde{S}_j^i$ are the j-th index of S^i and $\tilde{S}^i$, respectively. Then, the encoding of x is given by $\mathsf{TCenc}(x) = S$; see Figure 3.1.

It is easy to see that this encoding is online and that the alphabet is of polynomial size $|\Sigma|^{O(\log n)} = \mathrm{poly}_{d,\delta}(n)$. Next we show that the encoding satisfies the distance property of a tree code (Definition 2.1).

Let $x, y \in [d]^m$ be two strings of the same length $m \leq n$ that differ starting at their j-th index (i.e., $x_1 \cdots x_{j-1} = y_1 \cdots y_{j-1}$ and $x_j \neq y_j$). For the analysis we will consider only long-enough divergent paths in which $m - j + 1 \geq 16$. The distance requirement for short diverging paths can be easily satisfied by, for example, finding a tree code $\mathcal{T}_{32}$ of depth 32 and distance δ by exhaustive search (this takes a constant time) and concatenating a copy of $\mathcal{T}_{32}$, every 16 levels: set

$$\begin{aligned}
S_{\text{short}}(x) &= \mathsf{TCenc}_{\mathcal{T}_{32}}(x_1 \cdots x_{32}) \circ \cdots \circ \mathsf{TCenc}_{\mathcal{T}_{32}}(x_{n-31} \cdots x_n) \\
S_{\text{short shift}}(x) &= 0^{16} \circ \mathsf{TCenc}_{\mathcal{T}_{32}}(x_{16} \cdots x_{47}) \circ \cdots \circ \mathsf{TCenc}_{\mathcal{T}_{32}}(x_{n-47} \cdots x_{n-16}) \circ 0^{16},
\end{aligned}$$

and concatenate to each index of S_i the corresponding symbol in $(S_{\text{short}}(x), S_{\text{short shift}}(x))$. This increases the size of the alphabet by only a constant.

Let $i^* = \lfloor \log(m - j + 1) \rfloor$ be the largest power of 2 that is at most $m - j + 1$. We can assume $i^* \geq 3$. Let us also write $j = t \cdot 2^{i^*} - r_j$ and $m = (t \cdot 2^{i^*} + 1) + r_m$ with some $t, r_j, r_m \in \mathbb{N}$ and $0 \leq r_j, r_m < 2^{i^*}$. First we show that if the endpoint m is aligned with 2^{i^*}, then there is a sequence of C_i with $0 \leq i \leq i^*$ that exactly covers the interval $[j, m]$ and provides a relative distance of δ in this case.

Lemma 3.2. If $m \mod 2^{i^*} = 0$, then $\Delta(S(x), S(y)) \geq \delta(m - j + 1)$.

Proof. The claim holds by induction on i^*. For $i^* < 5$ the claim holds due to $S_{short}, S_{short\text{-}shift}$. For the induction step, note that indices $[m - 2^{i^*} + 1, m]$ in S^{i^*} consists of the codeword $C_{i^*}(x_{m-2\cdot 2^{i^*}+1} \cdots x_{m-2^{i^*}})$ and note that $j \in [m - 2 \cdot 2^{i^*} + 1, m - 2^{i^*}]$. It follows that restricted

to the indices $[m-2^{i^*}+1, m]$, the distance between $S(x)$ and $S(y)$ is at least $\delta 2^{i^*}$. By the induction hypothesis on $x_1 \cdots x_{m'}, y_1 \cdots y_{m'}$ with $m' = m - 2^{i^*}$ (which satisfies $m' \mod 2^{i^*-1} = 0$), we find that restricting to the indices $[j, m']$, the distance is at least $\delta(m' - j + 1)$; thus

$$\Delta(S(x), S(y)) \geq \delta 2^{i^*} + \delta(m - 2^{i^*} - j + 1) \geq \delta(m - j + 1). \qquad \square$$

We now prove the case where $m \mod 2^{i^*} \neq 0$. We create two intervals: $[j, t2^{i^*}]$ and $[t2^{i^*}+1, m]$. In the first interval, $[j, t2^{i^*}]$, the endpoint is aligned with 2^{i^*-1}, and using the above lemma we get that the distance provided restricted to these indices is at least $\delta(t2^{i^*} - j + 1) = \delta(r_j + 1)$.

As for the interval $[t2^{i^*}+1, m]$, we distinguish two cases. First note that $r_m + r_j + 2 > 2^{i^*-1}$. Thus, one of r_j, r_m is at least $2^{i^*-1} - 1$. If $r_m \geq 2^{i^*-1} - 1$, we use the codeword $C_{2^{i^*}}(x_{(t-1)2^{i^*}+1} \cdots x_{t2^{i^*}})$, which lies along the indices $[t2^{i^*}+1, (t+1)2^{i^*}]$ in S^{i^*}. Since $j \in [(t-1)2^{i^*}+1, t2^{i^*}]$, this codeword provides us with at least $\max\{0, \delta 2^{i^*} - (2^{i^*} - 1 - r_m)\}$ different indices in the interval $[t2^{i^*}+1, m]$, and since $r_m \geq 2^{i^*-1} - 1$, the relative distance in the interval $[j, m]$ in this case is at least

$$\begin{aligned}
\frac{\delta(r_j + 1) + \max\{0, \delta 2^{i^*} - (2^{i^*} - 1 - r_m)\}}{r_m + r_j + 2} &\geq \frac{\delta(r_j + 1) + \delta 2^{i^*} - 2^{i^*-1}}{r_j + 2^{i^*-1} + 1} \\
&\geq \frac{\delta + \delta 2^{i^*} - 2^{i^*-1}}{2^{i^*-1} + 1} \\
&\geq \frac{9\delta - 4}{5},
\end{aligned}$$

where the last transition occurs because we assume $i^* \geq 3$.

The other case is when $r_j \geq 2^{i^*-1} - 1$, and thus $r_m \leq 2^{i^*-1} - 1$. Here we use the shifted string $\tilde{S}^{i^*-1}$: the codeword $C_{i^*-1}(x_{(t-1)2^{i^*}+1} \cdots x_{t2^{i^*}-2^{i^*-1}})$, which encodes x_j, lies along the indices $[t2^{i^*}+1, t2^{i^*}+2^{i^*-1}]$, and gives us a distance of $\max\{0, \delta 2^{i^*-1} - (t2^{i^*} + 2^{i^*-1} - m)\}$. The relative distance in $[j, m]$ is then at least

$$\begin{aligned}
\frac{\delta(r_j + 1) + \max\{0, r_m - 2^{i^*-1}(1-\delta) + 1\}}{r_j + r_m + 2} &\geq \frac{\delta 2^{i^*-1}}{2^{i^*-1} + 2^{i^*-1}(1-\delta)} \\
&\geq \frac{\delta}{2 - \delta}.
\end{aligned}$$

To summarize, the relative distance in any $[j, m]$ is at least $\min\{\frac{9\delta-4}{5}, \frac{\delta}{2-\delta}, \delta\}$. For any $\alpha < 1$, the preceding construction with $\delta \geq \frac{5\alpha+4}{9}$ gives a d-ary tree code of depth n, distance α, and alphabet size $\left(\frac{c \cdot d}{1-\alpha}\right)^{1+2\log n}$, for some constant c. For any constants d, α, the alphabet size is $\text{poly}_{\alpha,d}(n)$. $\square$

3.1.2 Coding schemes with a reduced success probability

If one is willing to compromise requiring a success probability of $1 - 2^{-\Omega(n_0)}$, then it is fairly easy to obtain an efficient coding scheme that has a constant rate and polynomially small failure probability for random noise. A prominent approach towards this goal is to replace the tree code with the following relaxation:

Definition 3.1 (k-local tree code)**.** A d-ary *k-local* tree code over an alphabet Σ with distance parameter α is an infinite rooted d-ary tree in which each edge e is marked with a label $w_e \in \Sigma$ and

for any rooted paths $p_1 = (e_1, \ldots, e_n)$ and $p_2 = (e'_1, \ldots, e'_n)$ of the same length, that differ starting from the j-th level (i.e., $e_i = e'_i$ for all $i < j$, but $e_j \neq e'_j$) and for which $n - j < k$, it holds that

$$\Delta\left(label(p_1), label(p_2)\right) \geq \alpha(n - j + 1).$$

That is, local tree codes satisfy the distance property only for "short" divergent paths, where the length of the diverging parts is at most k.

Using a k-local tree, we can obtain a coding scheme with failure probability $< 2^{\log n - O(k)}$ by performing Algorithm 1 replacing the tree code with a k-local tree code. The intuition is that if we consider k consecutive transmissions, the local tree code behaves like a standard (i.e., nonlocal) tree as long as the number of errors in that chunk is small enough, say smaller than ck for some small-enough constant c. On the other hand, the probability of having a stretch of k consecutive transmissions with more than ck bit corruptions anywhere throughout the protocol is bounded by

$$n \cdot \varepsilon^{ck} \leq 2^{\log n - c'k},$$

which is polynomially small for $k = O(\log n)$.

Schulman [77] observed that $O(\log n)$-local tree codes can be efficiently constructed in $\mathrm{poly}(n)$ time: one can find a tree code $\mathcal{T}$ of depth $2c \log n$ via exhaustive search, and then concatenate the same trees every $c \cdot \log n$ layers. That is, each label in the local tree is composed of two labels of $\mathcal{T}$, where the label of the last edge in the path $p = (e_1, \ldots, e_n)$ is computed the following way. Let $k = c \log n$ and assume $n = t \cdot k + r$ where $t, r \in \mathbb{N}$ and $r < k$. Let $p_{-1} = (e_{tk}, \ldots, e_n)$ and $p_{-2} = (e_{(t-1)k}, \ldots, e_n)$. Then, the label of the last edge in p is given by $(\mathsf{TCenc}_{\mathcal{T}}(p_{-1}), \mathsf{TCenc}_{\mathcal{T}}(p_{-2}))$.

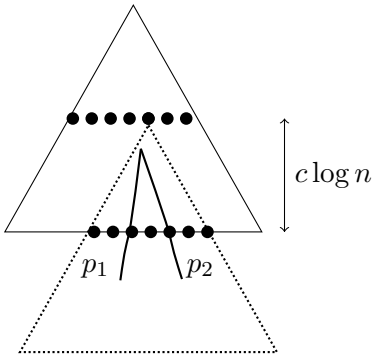

Figure 3.2: A concatenated tree code. Every black dot represents a concatenation of another tree code $\mathcal{T}$ of depth $2c \log n$ (only one such concatenation is depicted). Every two diverging paths p_1, p_2 of length $\leq c \log n$ are fully contained within at least one of the concatenated trees.

Every two divergent paths of length $k \leq c \log n$ are fully included within a *single* $\mathcal{T}$, and thus their corresponding labelings must satisfy the distance property; see Figure 3.2.

Theorem 3.3. For any constant $\varepsilon < 1/2$, there exists a randomized efficient coding scheme that simulates π_0 over a BSC_ε, has a constant rate, and has a success probability of $1 - \mathrm{poly}(1/n_0)$.

3.1.3 Further Reading: Other efficient schemes with reduced success probabilities

In this section we review other coding schemes that obtain computational efficiency by sacrificing the success probability.

Tree code based schemes

A comprehensive analysis of the tree-based coding scheme of [76] when replacing the tree code with a local-tree relaxation appears in [65]. A formal statement of the construction of §3.1.2 also appears in [14]. Moitra [65] showed another technique for efficiently constructing $O(\log n)$-local tree codes based on high-girth expander graphs.

Braverman [14] showed how to construct a $O((\log n)^c)$-local tree of depth n for any $c > 1$ in time $2^{O(\log n)^{1/c}}$, that is, in subpolynomial time in n. Encoding and decoding takes a similar amount of time. This leads to an efficient coding scheme in the random noise setting with a success probability of $1 - 2^{-\Omega((\log n_0)^c)}$. Furthermore, the obtained scheme is deterministic.

Peczarski [70] showed a randomized construction for a tree code with distance parameter $\alpha = 1/2$ that uses an alphabet of size $O(\exp(\sqrt{\log 1/\varepsilon}))$ and succeeds with probability $1 - \varepsilon$. This gives a tradeoff between the rate $O(1/\sqrt{\log 1/\varepsilon})$ and the success probability $1 - \varepsilon$ of the obtained coding scheme, assuming random noise. Specifically, obtaining coding schemes with success probability of $1 - 2^{-\Omega(n_0)}$ incurs a vanishing rate of $O(1/\sqrt{n_0})$, while a reduced success probability of $1 - 1/\text{poly}(n_0)$ gives a vanishing rate of $O(1/\sqrt{\log n_0})$.

Rewind-if-error based schemes

The scheme that gets closest to the desired exponential success probability is a scheme by Schulman [74], which follows a rewind-if-error paradigm where, for every $i \geq 1$, after 2^i steps of the noiseless protocol the parties exchange $O(2^i/i^2)$ checking bits that verify the transcript so far, and rewind if error has been found. The rewinding mechanism allows the simulation to revert back to two possible meeting points (i.e., rewind the protocol to nearest multiple of 2^i or to the previous multiple of 2^i). We will explore these ideas further in §5 when discussing communication-efficient coding schemes. The scheme of [74] provides the following.

Theorem 3.4 ([74])**.** For any constant $\varepsilon < 1/2$, there exists a randomized efficient coding scheme that simulates π_0 over a BSC_ε, has a constant rate, and has a success probability of $1 - 2^{-\Omega(n_0/\log^4 n_0)}$.

3.2 Efficient coding schemes over BSC: Potent tree codes

The first efficient scheme over BSC channels that achieves constant rate and exponentially small failure probability was suggested by Gelles, Moitra, and Sahai [41, 42]. As with local trees, the main idea is to replace the inefficient tree code with a relaxed notion called *potent tree code*. The relaxed structure is a labeled tree in which most of the divergent paths, but not all of them, satisfy the distance properties. The fraction of paths that do not satisfy the distance property may be large; however, only a small number of them collide with any given root-to-leaf path.

Definition 3.2 (potent tree code)**.** An (ε, α)-potent tree code is a d-ary tree of depth n whose edges are labeled with symbols from Σ. For any root-to-leaf path P, the following holds.

- Two paths $p \circ p_1$ and $p \circ p_2$ of the same length are called *bad paths* of length ℓ if $|p_1| = |p_2| = \ell$, the first edge in p_1 differs from the one in p_2, and

$$\Delta(label(p \circ p_1), label(p \circ p_2)) < \alpha\ell.$$

  The divergent suffices p_1, p_2 are called bad subpaths.

- The number of nodes along P that belong to a bad subpath is at most εn.

Theorem 3.5 ([43, 42])**.** For any constants $\varepsilon, \alpha < 1$, a random labeling of a d-ary tree of depth n from alphabet of size $|\Sigma| = d^{O_{\varepsilon,\alpha}(1)}$ yields a (ε, α)-potent tree with probability $1 - 2^{-\Omega(n)}$.

Proof. Assume $\mathcal{T}$ is a d-ary tree of depth n, labeled uniformly and independently from an alphabet Σ whose size we determine later. We bound the probability that $\mathcal{T}$ fails to be (ε, α)-potent.

If $\mathcal{T}$ is not (ε, α)-potent, there must exist a path P and t bad paths of length $\ell_1, \ldots, \ell_t$ along P, such that $\sum \ell_i > \varepsilon n$. We can select a subset of these bad paths so that the selected paths satisfy

1. the divergent part (along P) for each two bad paths is edge-disjoint;
2. the total length of the paths is $\geq \varepsilon n/2$.

This is immediate from the following lemma.

Lemma 3.6 ([76])**.** Let $\ell_1, \ell_2, \ldots, \ell_n$ be intervals on $\mathbb{N}$, whose union is of size σ. Then there exists a subset $S \subseteq \{1, 2, \ldots, n\}$ such that for any $i, j \in S$, $\ell_i \cap \ell_j = \emptyset$, and $\sum_{i \in S} |\ell_i| \geq \sigma/2$.

The probability of having a bad path of length ℓ, say, between levels $[h, h + \ell]$ in P is the probability that some path that diverges from P at the h-th level will have labels that collide with the labels along P in the interval $[h, h + \ell]$ by a fraction of at least $(1 - \alpha)$. Summing over all such d^ℓ divergent paths, we bound this probability by

$$d^\ell \sum_{x=(1-\alpha)\ell}^{\ell} \binom{\ell}{x} \left(\frac{1}{|\Sigma|}\right)^x \leq d^\ell 2^\ell |\Sigma|^{-(1-\alpha)\ell},$$

which with the appropriate choice of $|\Sigma|$, can be smaller than, say, $d^{-4\ell/\varepsilon}$. Then, the probability of having *disjoint* bad paths along P, whose total length is at least $\varepsilon n/2$, is bounded by

$$\prod_i d^{4\ell_i/\varepsilon} = d^{(4/\varepsilon)\sum_i \ell_i} \leq d^{-2n}.$$

Note that since the bad paths we consider are disjoint, the probability of a certain (sub)path being bad is independent of the other (sub)paths. Using a union bound over all d^n possible paths P we complete the proof and conclude that $|\Sigma| = d^{O(1/\varepsilon(1-\alpha))} = d^{O_{\varepsilon,\alpha}(1)}$. □

The main observation of [42] is that potent tree codes are good enough to replace tree codes in coding scheme for interactive communication. Qualitatively, a bad node of the potent tree is a node for which the code has no distance guarantees, and therefore it is probable that the decoding will fail to output the correct node at those places. However, since the fraction of the bad nodes along any path is at most ε, this amounts to at most εn times where the potent tree code fails or, equivalently, additional εn rounds where the simulation may not decode the correct sent word (even if there were no errors). Practically, this can be compared to a simulation with a standard tree code that observes a fraction of noise that is higher by an *additive* factor of at most ε. Thus, if the scheme (assuming a standard tree code) is resilient to a fraction c of adversarial noise, it becomes resilient to a fraction $c - \varepsilon$ of adversarial noise when replacing the tree codes with potent tree codes. For random noise, this amounts to having a worse noise parameter, yet it keeps the scheme resilient for any noise parameter below $1/2$ (by choosing a potent code with the right parameters).

The last step that is needed in order to obtain an explicit coding scheme construction, is to make both parties agree on the potent tree in use. Since the construction is random, the parties cannot construct the trees independently (their trees will mismatch), and since it takes $O(d^n)$ to describe the tree, if one party constructs it and sends it to the other side, the rate will be vanishing.

Luckily, it is possible to partially derandomize the construction of potent tree codes [42] using ε-biased sample spaces [3, 67]. To simulate a protocol of length n_0, one party samples a random seed of length $S = O(n_0)$ bits and sends it to the other side encoded via some error-correction code. Then both parties generate a string of length d^S, which is almost 2^S-wise independent. This string is used to label the potent tree in a straightforward way. The 2^S-wise independence guarantees that the labels of any two root-to-leaf paths in the tree are almost independent, which is enough to show that the tree is potent with high probability. The parties then continue with a tree code based coding scheme, e.g., Algorithm 1, and obtain an efficient coding scheme that succeeds with probability $1 - 2^{-\Omega(n_0)}$, assuming a BSC channel. Theorem 3.7 summarizes these ideas.

Theorem 3.7 ([41, 42])**.** For any constant $\varepsilon < 1/2$, there exists a randomized efficient coding scheme that simulates π_0 over a BSC_ε and has a constant rate and a success probability of $1 - 2^{-\Omega(n_0)}$.

3.3 Efficient coding schemes for adversarial noise: Suboptimal noise resilience

We now focus on the case of adversarial noise. Recall that no efficient decoding of tree codes is known, assuming a constant fraction of adversarial noise.

The first efficient coding scheme in this setting was suggested by Brakerski and Kalai [10] and later improved upon by Brakerski and Naor [12] to obtain almost linear time; see also [11]. The idea behind the coding scheme is to split the simulation into chunks of logarithmic size. Each chunk is simulated via a inefficient scheme (say, Algorithm 1). The synchronization *between* chunks, that is, choosing which chunk is the next to be simulated, is performed using a rewind-if-error paradigm similar to Algorithm 2, that is, by exchanging hashes of the transcripts simulated so far, and proceeding to the next chunk if the hash values match (or otherwise, rewinding to a previous chunk). To protect the synchronization part from noise, we can assume that each simulated chunk begins with exchanging the hash values and only then proceeds to transmissions of π_0, while the entire chunk (including the synchronization parts) is being simulated via Algorithm 1.

Since each chunk is of logarithmic size, Algorithm 1 takes $\exp(c \log n_0) = \mathrm{poly}(n_0)$ time per chunk; computing and comparing hashes, and rewinding accordingly can be done efficiently. This yields an efficient scheme that is resilient to a fraction of $1/16 - \varepsilon$ corruptions, assuming a large yet constant alphabet.[1] The scheme is described in Algorithm 3.

Theorem 3.8. Algorithm 3 is efficient, has a constant rate, and is resilient to $1/16 - \varepsilon$ symbol corruptions with probability $1 - 2^{-\Omega(n_0)}$.

Proof. The efficiency and rate claims are immediate. We show that as long as the adversarial noise level is below $1/16 - \varepsilon$, the algorithm succeeds in simulating π_0.

We say that a chunk is *good* if the output (Line 9) is correct: each party receives the correct (hash) values s, l, h sent by the other side, and the simulated transcript $\tilde{T}$ is equal to the noiseless transcript of the same chunk. Otherwise, we say that the chunk is bad.

[1] Similarly, the scheme is resilient to up to $1/32 - \varepsilon$ bit flips.

Algorithm 3 The Brakerski-Kalai coding scheme [10]

Input: a protocol π_0 of length $n_0 = |\pi_0|$ and an input x; a noise-resilience parameter $1/16 - \varepsilon$.

Let $\mathcal{H} = \{h_s : \{0,1\}^{c \cdot n_0} \to \{0,1\}^{\log n_0}\}$ be a hash function family given by Theorem 2.8, with seed size $|s| = \log n_0$.

$k = \log n_0$
$T \leftarrow \emptyset$
for $i = 1$ to $N = n_0/\varepsilon k$ **do**
    run Algorithm 1 on the noiseless protocol π_T defined as follows:
    procedure π_T:
        sample s and send $(s, |T|, h_s(T))$
        run π_0 for k rounds (assuming the past is T)
    end procedure

    let $(\tilde{s}, \tilde{l}, \tilde{h}), \tilde{T}$ be the output of the simulation of π_T, with $|\tilde{T}| = k$
    if $|T| = \tilde{l}$ and $h_{\tilde{s}}(T) = \tilde{h}$ **then**
        $T \leftarrow T \circ \tilde{T}$
    else
        if $|T| \geq \tilde{l}$, delete k bits from the suffix of T
output T (a prefix of length n_0)

Next, we use a similar potential function argument as in the proof of Algorithm 2. Define the potential function $\varphi(i)$ as the length of longest correct prefix in T_A and T_B at the end of iteration i, minus the length of the incorrect suffixes—let $m(i)$ be the maximal point where both $T_A(i), T_B(i)$ equal T_0 (in blocks of size k). Then

$$\varphi(i) = 3\left\lfloor \frac{m(i)}{k} \right\rfloor - \frac{|T_A(i)| + |T_B(i)|}{k}.$$

Using the same argument as in Algorithm 2, for each good iteration i in which there is no hash collision, we have $\varphi(i) \geq \varphi(i-1) + 1$, while for each bad iteration or when there are hash collisions, $\varphi(i) \geq \varphi(i-1) - 3$.

Next, we bound the number of hash collisions throughout the protocol. The probability of having a hash collision, given that the seed is uniformly chosen, is $2^{-\log n_0} \leq 1/n_0$; thus the expected number of hash collisions in the simulation is at most $N \cdot 1/n_0 = 1/\varepsilon k$. The probability that there were more than εN hash collisions throughout the protocol[2] is bounded via a tail bound by

$$\begin{aligned}
\Pr[\#\text{collisions} > \varepsilon N] &= \Pr\left[\#\text{collisions} > (\varepsilon N \cdot \varepsilon k) \cdot \frac{1}{\varepsilon k}\right] \\
&\leq \exp\left(-\Omega\left(\frac{n_0^2}{k}\right)\right) \\
&\leq 2^{-\Omega(n_0)}.
\end{aligned}$$

We can therefore assume that the potential has decreased due to hash collisions in at most εN iterations, with probability $1 - 2^{-\Omega(n_0)}$.

[2]We note that hash collision decreases the potential only if the parties are unsynchronized, and has no effect if the parties are synchronized. We nevertheless can assume that every collision is harmful, and the proof still holds.

If at the end of the simulation, $\varphi(N) < n_0/k$, then there must have been at least $(1/4 - \varepsilon)N$ iterations in which the potential has not increased. Recalling that the number of iterations with hash collisions is bounded by εN, we see that the fraction of bad iterations must be at least $(1/4 - 2\varepsilon)$. Since each chunk is simulated via Algorithm 1, if the output in Line 9 is incorrect, the fraction of errors during the simulation of that chunk must be at least $1/4 - \varepsilon$. Therefore, the global noise fraction in this failed simulation must be at least

$$\frac{(1/4 - \varepsilon) \cdot (1/4 - 2\varepsilon)N}{N} = \frac{1}{16} - O(\varepsilon). \qquad \square$$

3.4 Efficient coding schemes for adversarial noise: Optimal noise resilience

The drawback of the efficient scheme described in the last section is that it can tolerate only up to $1/16 - \varepsilon$ adversarial noise, while the optimal bound on the adversarial noise is $1/4$, as discussed in §2. In this section we describe a scheme by Ghaffari and Haeupler [45] that is both efficient and resilient to the optimal noise fraction of $1/4 - \varepsilon$. To that end, we need to detour first through a new concept of coding for interactive communication, namely, *interactive list-decoding*.

3.4.1 Interactive list-decoding coding schemes

In an equivalent manner to list-decoding codes in the one-way communication setting [31, 82], we can define interactive list-decoding as an interactive protocol that outputs a small list of possible outcomes, so that as long as the noise is small enough, the correct output $f(x, y)$ appears in the list [46, 45, 15].

Definition 3.3 (interactive list-decoding coding scheme)**.** An interactive list-decoding protocol for a function f with list size L and noise resilience ρ is an interactive protocol in which both parties output a list of L values, and if the noise rate is at most ρ, then $f(x, y)$ appears in the output list of both parties.

Braverman and Efremenko [15] showed how to construct an interactive list-decoding coding scheme by augmenting the notion of tree codes into *list tree codes*. When decoding a codeword x of length n via a list tree code, one gets a list of size at most Ln that contains all prefixes of messages y so that $\Delta_{\mathsf{sfx}}(\mathsf{TCenc}(y), x) < \alpha$. This implies that for every length $n' < n$, the list tree code provides a list $L_{n'}$ of messages of length n' that may have been encoded, and the size of this list is L *on average*, that is, $\sum_{n'<n} \frac{1}{n}|L_{n'}| \leq \frac{1}{n} \cdot Ln = L$. Formally, we define list tree codes as follows.

Definition 3.4 (list tree codes)**.** An (α, L)-list tree code $\mathcal{T}$ is a d-ary tree of depth n labeled with symbols from Σ. The labeling of the tree satisfies the following. For any word w of length n, let L_i be the list of codewords x of length i that are close to a prefix of length i of w in their suffix distance, that is,

$$L_i = \{x_1 \cdots x_i \mid \Delta_{\mathsf{sfx}}(\mathsf{TCenc}_{\mathcal{T}}(x_1 \cdots x_i), w_1 \cdots w_i) < \alpha\}.$$

Then it holds that $\sum_{i=1}^{n} |L_i| \leq n \cdot L$.

Similar to the case of tree codes, no efficient deterministic construction for list tree codes exists. However, Braverman and Efremenko show that a random labeling of the tree yields a list tree code with overwhelming probability.

Theorem 3.9 ([15]). For any $0 < \alpha < 1$ and $d > 0$, a d-ary tree of depth n in which each edge is labeled uniformly from an alphabet Σ with $|\Sigma| > (2d)^{3/\alpha^2}$, is an (α, L)-list tree code with $L = \frac{1}{\alpha} + 1$ with probability at least $1 - 2^{-n}$.

On the surface, using a list tree code does not seem so useful for obtaining interactive coding schemes with a good rate: if at every round where we decode the tree, we obtain L possible decodings, then after n_0 rounds, the number of possible answers explodes to L^{n_0}. However, a more carful analysis shows this is not the case.

Assume a variant of Algorithm 1 in which each symbol of Σ encodes L possible edges (instead of only a single edge). Then, whenever Alice decodes, at some round i, all the symbols she has received so far, she gets (on average) L strings of length i of symbols of Σ. These amount to L *sets* of edges Alice should consider and extend (rather than a single one, as in Algorithm 1). Each such set, when combined with Alice's own edges, either leads to a unique root-to-leaf path, or is clearly an incorrect decoding of Bob's edge set. Therefore, Alice has (on average) at most L potential paths to consider at every round. Then, Alice extends each one of these L paths by a single edge from the set of her own edges. Note that a *single* symbol of Σ suffices to describe these new L edges that Alice wishes to send on the next round. Thus, the number of possible answers remains L and doesn't explode exponentially.

Combining list tree codes with the simulation ideas of Algorithm 1 leads to the following.

Theorem 3.10 ([15]). For any $\varepsilon > 0$, there exists an interactive list-decoding scheme that simulates any π_0 in $O_\varepsilon(n_0)$ rounds, sends symbols from alphabet of size $O_\varepsilon(1)$, outputs a list of size $O_\varepsilon(1)$, and is resilient to $1/2 - \varepsilon$ fraction of corrupted symbols.

We note that the output of the preceding coding scheme is in fact a path on the underlying tree that describes the noiseless protocol (see §2.1.3) rather than the final output $f(x, y)$. This allows the parties to verify that the output is fully consistent with their own input. We call a list-decoding scheme with this property a *verifiable* list-decoding scheme. This verifiability property will be very useful when using list decoding as a primitive in order to obtain a unique-decoding coding scheme with optimal resilience.

3.4.2 From list decoding to unique decoding

List decoding is a simpler task than unique decoding. On the other hand, it allows a greater noise resilience. In this section we describe how to reduce unique decoding to list decoding, that is, how to obtain an interactive coding scheme by using a list-decoding coding scheme. Most of the techniques presented in this section appeared in [45].

Theorem 3.11 ([45]). For any $\varepsilon > 0$ for which there exists a verifiable list-decoding scheme with noise resilience $1/2 - \varepsilon$ and rate $\Theta_\varepsilon(1)$, there exists a (unique-decoding) coding scheme with noise resilience $1/4 - \varepsilon$ and rate $\Theta_\varepsilon(1)$, with essentially the same computational complexity.

Proof. Consider the coding scheme sketched in Figure 3.3: given the noiseless protocol π_0, we sequentially perform $N = 1/\varepsilon$ independent instances of the list-decoding scheme simulating π_0. Each such instance takes $n = O_\varepsilon(n_0)$ rounds over an alphabet of size $|\Sigma_1| = O_\varepsilon(1)$, and, if the noise fraction during that instance is less than $1/2 - \varepsilon$, the output is a list of at most L paths of length n_0 in the tree $\mathcal{T}_0$ describing the noiseless protocol π_0.

Assume a verifiable list-decoding scheme π of length n and resilience $1/2 - \varepsilon$.

A unique-decoding coding scheme for noiseless protocol π_0:
For $i = 1$ to $N = 1/\varepsilon$, repeat:

1. Simulate π_0 using the verifiable list-decoding scheme and obtain output list Y_i (remove invalid outputs from the list).
2. Parallel to Step 1, send $S_i = \cup_{j<i} Y_j$ to the other side encoded with an error-correcting code with relative distance $1 - \varepsilon$, that is, send $send = \mathsf{ECC}(S_i)$.
3. Receive the codeword *recv* from the other side, and decode $\tilde{S}_{i-1} = \mathsf{DEC}(recv)$. Assign each consistent path in $\tilde{S}_{i-1}$ with confidence $c_i = 1 - \frac{2\Delta(recv, \mathsf{ECC}(\tilde{S}_{i-1}))}{n}$.

Output the valid path that has gained the most confidence.

Figure 3.3: From a list-decoding scheme to a unique-decoding scheme

Parallel to the preceding process, at iteration i we also communicate the list of paths S_i obtained during the simulations of iterations less than i (removing any invalid path, i.e., path that is inconsistent with the input of that party). To that end, first note that a list of L paths of length n_0 in T_0 takes at most $O(Ln_0)$ bits to communicate. Thus, for any i, we can encode S_i using $O(NLn_0) = O_\varepsilon(n_0)$ bits. Next, we encode each such S_i using a code ECC over an alphabet of size $|\Sigma_2| = O_\varepsilon(1)$, with relative distance $1 - \varepsilon$; such a code with length $O_\varepsilon(n_0)$ can be easily found, and we will assume its length is exactly n (increasing the length of the coding scheme π if needed). Since both π and ECC are of the same length, we can concatenate each symbol of the codeword to a symbol of the list-decoding scheme. This is done by using a larger alphabet $\Sigma = \Sigma_1 \times \Sigma_2$, which is still of constant size, without increasing the number of rounds the coding scheme takes.

The output is based only on the sets of paths $\tilde{S}_i$ the party receives. After receiving a specific S_i, the receiver removes any path that is inconsistent with its input, and gives a weighted vote to each of the remaining paths. The weight of the vote (the *confidence*) is given according to the estimated noise in the received word, *recv*: The closer *recv* is to a valid codeword, the higher the confidence of the path in $\tilde{S}_i = \mathsf{DEC}(recv)$. The output of the entire scheme is the path that has received the highest overall confidence.

The analysis idea is simple: if at some iteration there were less than $(1 - \varepsilon)/2$ corruptions, then the correct path will be included in the list S_i of that iteration and all subsequent iterations. Then, at every subsequent iteration that has at most $(1 - \varepsilon)/2$ errors, the receiver adds confidence to the correct path. If the total noise level is below $1/4 - \varepsilon$ errors, then the correct path will get a positive confidence. Furthermore, note that as long as the set S_i is decoded correctly, no incorrect value gains confidence (since the sender sends only outputs that are consistent with his input, and the receiver eliminates any output inconsistent with its own input). We now formally compute the confidence of the correct path and show it surpasses the confidence of any incorrect path.

Assume that i^* is the first iteration in which the list decoding outputs the correct path. Since π failed in all iterations less than i^*, it must be that the relative noise fraction during these iterations is at least $(1 - \varepsilon)/2$. Let ρ_i be the noise fraction of the codeword *recv* (without loss of generality, received by Alice) at instance i. Consider the two cases where (1) $\tilde{S}_i = S_i$, so the confidence is $c_i = 1 - 2\rho_i$, and (2) $\tilde{S}_i \neq S_i$, that is, when $\rho_i > (1 - \varepsilon)/2$, so the confidence is $c_i = 2\rho_i - (1 - 2\varepsilon)$.

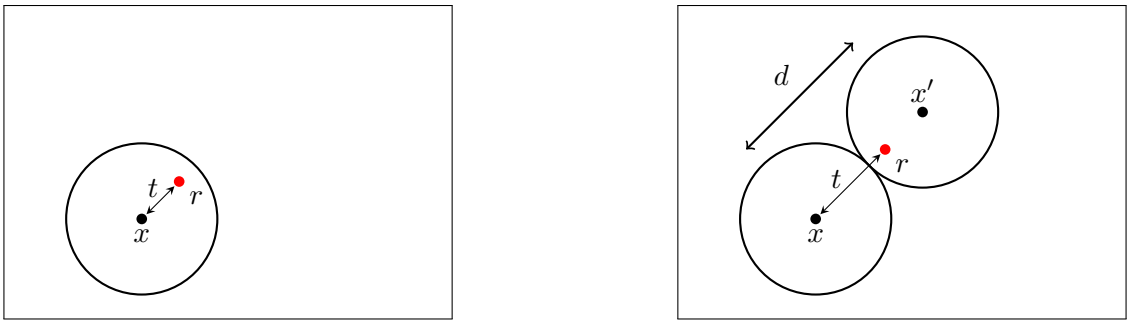

Figure 3.4: An illustration of the two types or errors. Both x and x' are codewords with distance d, x is the sent word, r is the received word, and t is the noise. In the first case, $t = \Delta(x, r)$; in the second, $t = d - \Delta(r, x')$.

See Figure 3.4.

Let $I_g \subseteq [N]$ denote all the iterations in which the correct path appears in Bob's S_i and Alice correctly decodes the received word $\tilde{S}_i = S_i$. Then incorrect paths gain confidence only in iterations not in I_g. We need to show that $\sum_{I_g} c_i > \sum_{\overline{I_g}} c_i$. Also note that in all the iterations in $\overline{I_g}$, it holds that $\rho_i > (1-\varepsilon)/2$: otherwise, if $i < i^*$, then the list decoding would have succeeded, and if $i > i^*$, then the decoding of S_i must be successful, and the iteration must belong in I_g instead. We have that

$$\begin{aligned}
\sum_{I_g} c_i - \sum_{\overline{I_g}} c_i &\geq \sum_{I_g}(1-2\rho_i) - \sum_{\overline{I_g}}(2\rho_i - (1-2\varepsilon)) \\
&\geq |I_g| - 2\sum_i \rho_i + (1-2\varepsilon)(N - |I_g|) \\
&> 2\varepsilon|I_g| + (1-2\varepsilon)N - 4\rho N \\
&\geq (1 - 2\varepsilon - 8\varepsilon^2)N - 4 \cdot (\tfrac{1}{4} - \varepsilon)N \\
&\geq \left(2\varepsilon - 8\varepsilon^2\right) N \\
&> 0,
\end{aligned}$$

where ρ is the global fraction of noise, $\rho = \frac{1}{2N}\sum_i \rho_i$. By noticing that if $\rho < 1/4 - \varepsilon$, then $|I_g| > 4\varepsilon N$: indeed, the maximal number of iterations that have $\rho_i > (1-\varepsilon)/2$ (at Alice side; assuming no noise at Bob's side) is below $(1-4\varepsilon)N$. This means that the correct path gets more confidence than all the other paths altogether, and thus the output must be correct. □

Using a slightly more careful analysis, Ghaffari and Haeupler [45] showed that there is no need to use an optimal list-decoding coding scheme that is resilient to a $\frac{1}{2}(1-\varepsilon)$ fraction of errors, but even a weaker list-decoding scheme that is resilient to a $\frac{1}{4}(1-\varepsilon)$ fraction of errors will do just fine.

Theorem 3.12 ([45])**.** For any $\varepsilon > 0$ for which there exists a verifiable list-decoding scheme with noise resilience $1/4 - \varepsilon$ and rate $\Theta_\varepsilon(1)$, there exists a (unique-decoding) coding scheme with noise resilience $1/4 - \varepsilon$, and rate $\Theta_\varepsilon(1)$, with essentially the same computational complexity.

3.4.3 Efficient coding schemes with optimal noise resilience

The final ingredient is obtaining an *efficient* list-decoding scheme that could be plugged into Theorem 3.11 and provide an efficient coding scheme that is resilient to the optimal fraction $1/4 - \varepsilon$ of

noise. Unfortunately, either the list-decoding scheme of Braverman and Efremenko (Theorem 3.10) or scheme of Braverman and Rao (Theorem 2.5) can take exponential time in the length of the simulated protocol. The key to obtaining an efficient list-decoding scheme is a *boosting* procedure given by Ghaffari and Haeupler [45], which practically augments the techniques of Brakerski and Kalai presented in §3.3 to the case of interactive list decoding. Namely, the boosting procedure simulates the protocol on small chunks (here, of size $\log^2 n_0$) and adds a layer of synchronizing between computation of chunks. However, since each chunk yields a list of possible transcripts, the synchronization part is more involved.

Theorem 3.13 (interactive list decoding boosting [45])**.** Assume a list-decoding scheme that simulates protocols of length $O(\log^2 \ell)$ in $O(R \log^2 \ell)$ rounds, outputs a list of size L, resilient to ρ fraction of noise with success probability $1 - 2^{-\Omega(\log^2 \ell)}$, and has a computational complexity T. Then, for any $\varepsilon > 0$, there exists a list-decoding scheme for noiseless protocols of length n_0 that takes $O(Rn_0)$ rounds, outputs a list of size $O(L/\varepsilon)$, resilient to a fraction $\rho - \varepsilon$ of noise, succeeds with probability $1 - 2^{-\Omega(n_0)}$, and has a computational complexity of $T \cdot \mathrm{poly}(n_0)$.

Applying the theorem onto the (exponential-time) protocol of Theorem 3.10 yields an efficient list-decoding scheme with resilience $1/2 - \varepsilon$ and constant list size $O_\varepsilon(1)$. Note that we need to apply the boosting at least twice[3] in order to simulate π_0 of length n_0 in an efficient way: the protocol of Theorem 3.10 is not efficient on chunks of size $O(\log^2 n_0)$. However, starting from chunk size $O((2 \log \log n_0)^2)$ and using Theorem 3.10, the boosting theorem guarantees an efficient list-decoding for protocols of length $O(\log^2 n_0)$ with good parameters, specifically, with a computational complexity of $2^{O(\log \log n_0)} \cdot \mathrm{poly}(\log n_0) = \mathrm{poly}(\log n_0)$. A second application of the boosting on that protocol yields the following.

Theorem 3.14 ([45])**.** For any $\varepsilon > 0$, there exists an efficient interactive coding scheme that simulates π_0 in $O_\varepsilon(n_0)$ rounds, is resilient to a fraction $1/4 - \varepsilon$ of noise, and succeeds with probability $1 - 2^{-\Omega(n_0)}$.

The boosting procedure is given in Algorithm 4. We now sketch the main ideas that lead to Theorem 3.13.

Theorem 3.13 proof sketch. Assume a protocol π_0 to simulate. The main idea is to cut the protocol into chunks of size $\log^2 n_0$ and run the list-decoding scheme guaranteed by the theorem separately on each chunk. As mentioned before, after each chunk the parties need to verify if they are synchronized and agree on the next chunk to be simulated. The difficulty here is that each party possesses a list of possible paths that previous chunks may have simulated (S_A for Alice and S_B for Bob). In order to decide which path is the correct one (so that the simulation will continue extending this part), the parties perform an interactive protocol that takes $O(\log^2 n_0)$ rounds, whose purpose is to determine the longest path both Alice and Bob have simulated so far.

The interactive algorithm that finds the longest common path in $S_A \cup S_B$ can be seen as a binary search over the set of paths S_A. Consider S_A as a set of edges. At each step Alice splits S_A into two parts by finding an edge e such that $|\mathcal{T}_e|$, the number of edges in S_A located in the subtree below e, is almost half of the entire set, that is, $|S_A|/3 \leq |T_e| \leq 2|S_A|/3$. Alice then sends Bob a hash value of the edge e. Bob replies with a single bit denoting whether the edge belongs to S_B or not. Alice

[3] A third application of the boosting theorem yields a coding scheme with almost linear computation time; see [45].

then recurses, either onto T_e or onto $S_A \setminus T_e$, both of which reduce the number of edges by at least one-third. The process completes after $O(\log_{2/3} n_0)$ rounds. We can assume that each hash value takes $O(\log n_0)$ bits, so this scheme takes $O(\log^2 n_0)$ rounds, assuming a constant-size alphabet. The scheme succeeds as long there is no hash collision between two different paths. The probability that two different paths cause a hash collision can be bounded by $O(N^2L^2) \cdot 2^{-\Omega(\log n_0)} = 2^{-\Omega(\log n_0)}$ by choosing a large-enough hash size. The failure probability can further be reduced to $2^{-\Omega(\log^2 n_0)}$; see [45]. □

Algorithm 4 Boosting interactive list-decoding schemes [45]

Input: a protocol π_0, visualized as the tree $\mathcal{T}_0$ of depth n_0.
Assume Alice's edges in $\mathcal{T}_0$ for the input x are given by the set E_x.

$S_A \leftarrow \emptyset$
for $i = 1$ to $N = O(\frac{1}{\varepsilon} n_0 / \log^2 n_0)$ **do**
    run an interactive protocol that finds the longest joint path between S_A and Bob's S_B.
    and let P be the output path
    run a list-decoding scheme on the next $O(\log^2 n_0)$ rounds of π_0 starting from the end of P,
    and assume Y_i is the set of output paths
    for each path $p \in Y_i$ **do**
        if p is consistent with E_x **then** $S_A \leftarrow S_A \cup p_j$
        if p leads to a leaf **then** add a vote to that leaf
output the $O(L/\varepsilon)$ leaves with most votes

Open Questions for Section 3

1. *Efficiently* construct a d-ary tree code of some distance α where the encoding of each label at depth of at most n takes $\text{poly}_{\alpha,d}(n)$ time to compute.
2. Design an efficient binary coding scheme that is resilient to the maximal possible fraction of noise.

4

Adaptive Coding Schemes

As we saw in previous sections, 1/4 is an upper bound on the fraction of noise any *nonadaptive* interactive protocol in which structure of the protocol (i.e., the order of speaking and the number of rounds) is predetermined and is independent of the noise pattern. In this section we discuss *adaptive* interactive protocols, which adaptively change their structure according to the observed noise. In particular, the identity of the party that sends the next symbol may depend on the transcript so far and, thus, on the noise. This adaptiveness allows protocols with noise resilience that surpasses the upper bound of 1/4 presented in Theorem 2.12.

Modeling adaptive protocols is a subtle task. Since the parties don't share the same view, they might have different beliefs regarding the identity of the next party to speak, and an adaptive model should carefully define how the protocol behaves in such situations. We discuss here two different approaches to model adaptive protocols. The first approach, by Ghaffari, Haeupler, and Sudan [46, 45], allows each party to decide, independently at each round, whether the party sends a symbol ("speaks") or awaits for an incoming symbol ("listens"), but it cannot do both. The second approach, by Agrawal, Gelles, and Sahai [1], allows parties to choose whether or not they *speak* at the next round; however, they always receive a sent symbol (i.e., they always *listen*).

Another issue that comes in adaptive protocols is determining the round of termination. While in nonadaptive protocols, the length of the protocol is fixed and predetermined, in adaptive protocol we may let each party decide on its own whether the protocol has terminated or not. Again, this may lead to instances where one party has terminated while the other has not, and the model should carefully define such situations. The approach of Agrawal et al. [1] allows also this sort of adaptivity.

4.1 Adaptive coding schemes: The speak-or-listen model

Consider the following model, in which each party decides, independently at each round, whether to *speak* or *listen*. In the nominal case, one party speaks and the other listens, so a symbol is transferred between the parties (unless corrupted by the channel). If both parties decide to speak, none of them receive the symbol sent by the other party (as none of them is expecting an incoming symbol). The

most subtle case is when both are set to listen (but no symbol is sent over the channel). In this case we allow the adversarial channel to determine which symbol they receive, *but this corruption is excluded from the noise count*, that is, in such rounds the error comes "for free". Except for this change, the setting behaves as in the standard model described in previous sections. Specifically, the simulation protocol π runs for n rounds, where up to a fraction ε of the symbols (in rounds where one party speaks and the other listens) may be corrupted by the adversarial channel.

This model, suggested by Ghaffari, Haeupler, and Sudan [46, 45], allows tolerating a maximal noise fraction of $2/7 - \varepsilon$. Note that $2/7 \approx 0.2857$ is larger than $1/4$; thus the resilience of these adaptive coding schemes outperforms the resilience of nonadaptive schemes.

Theorem 4.1 ([46]). For any $\varepsilon > 0$, there exists an adaptive coding scheme in the speak-or-listen model that simulates any π_0 in $O_\varepsilon(n_0)$ rounds, uses an alphabet of exponential size, and is resilient to a fraction $2/7 - \varepsilon$ of corrupted symbols.

Proof. The main idea is the following: split the protocol into seven chunks of equal size. The first six chunks are used to perform a standard (nonadaptive) coding scheme—for example, a simplified version of Algorithm 1 in which Alice maintains a set S_A (and Bob maintains S_B) that contains the edges Alice would have communicated in Algorithm 1. At each round Alice sends this entire set to Bob. Since the set contains only polynomially many edges, it can be described as a single symbol out of a polynomially large alphabet.

The last chunk is the adaptive part. Each party estimates the noise that it "sees" during the first six chunks and decides accordingly what to do during the last chunk: If the noise is high, it is probable that the information received from the other side is incorrect, and the party uses the last chunk to *listen*. If, on the other hand, the noise that party sees is low, then it is probable that the channel corrupted the communication going to the other side. Then, the party uses the last chunk to speak. The information in the chunk is all the information S_A (or S_B for Bob) accumulated during the first six chunks. The scheme is sketched in Algorithm 5.

Algorithm 5 Adaptive coding scheme with resilience 2/7 in the speak-or-listen model

Input: a protocol π_0 of length $n_0 = |\pi_0|$ and an input x; A noise-resilience parameter $2/7 - \varepsilon$.
Describe π_0 as the tree $\mathcal{T}_0$ and let E_x be the set of Alice's edges in $\mathcal{T}_0$, on input x. Set $n = n_0/\varepsilon$.

$S_A \leftarrow \emptyset$
for $i = 1$ to $3n/7$ **do**
    Send S_A, receive $\tilde{S}_B$ (takes 2 rounds)
    if $S_A \cup \tilde{S}_B$ has a unique rooted path that ends at Alice's owned node **then**
        $S_A \leftarrow S_A \cup e$ where e is the (unique) edge of E_x that extends that path
    if the unique path of $S_A \cup \tilde{S}_B$ reaches a leaf **then**
        add a vote to that leaf

let t be the total number of votes cast so far, and let s be the votes of the leaf with most votes
if $s > t - n/7$ **then**
    at each one of the next $n/7$ rounds, send S_A
else
    at each one of the next $n/7$ rounds, receive $\tilde{S}_B$; if $S_A \cup \tilde{S}_B$ leads to a leaf, add a vote to that leaf
output the path to the leaf that obtained the highest vote count

First, we note that each set S_A can be sent in a single round, using an alphabet of exponential size, where each symbol contains $O(n)$ bits. This follows since S_A is constructed one edge at a time, where each edge extends a previous path which is already contained in S_A; that is, S_A is a rooted subtree of size at most n, and can be fully described using $O(n)$ bits.

Next, we note that the number of votes the correct leaf gets at one side is at least $3n/7 - n_0 - B$, where B is the number of rounds that were corrupted during the first $6n/7$ rounds: during the first $6n/7$ rounds, only $3n/7$ are rounds in which a symbol is received and a vote may be cast (in the other $3n/7$ rounds, a symbol is *sent*, and a vote may be cast by the other party); furthermore, it takes n_0 rounds to "build" the path to the correct leaf so that in these rounds the correct leaf cannot get a vote. Since the channel is limited to making at most $(2/7-\varepsilon)n$ corruptions, the correct node gets at least $3n/7 - n_0 - (2n/7 - \varepsilon n) \geq n/7$ votes, while any incorrect node could get at most $(2/7 - \varepsilon)n$ votes. Therefore, if a certain node gets more than $t - n/7$ of the votes, where t is the total amount of votes cast by that party during the first $6n/7$ rounds of the protocol, that node must be the correct node.

Lemma 4.2. If a node v gets more than $t - n/7$ of the votes, then it is the correct node.

Proof. Assume towards contradiction that some wrong node $\tilde{v}$ gets s votes where $s > t - n/7$; the correct node v gets k votes, with $n/7 \leq k < s$; all the other nodes (together) get $p = t - s - k \geq 0$. Since $s > t - n/7$ we have $s > (s + k + p) - n/7$, thus $k < n/7 - p$, which is a contradiction. □

Furthermore, since any vote to an incorrect leaf (say) at Alice's side stems from a corrupt symbol sent to Alice, and since the total budget is less than $2/7$, for at least one party the correct node gets more votes. For that party, the correct node will get $s > n/7$ votes, while all other incorrect nodes will get at most $n/7$ votes together; thus $t - s \leq n/7$. Therefore, it is never the case that *both* parties are set to listen during the last $n/7$ rounds.

Finally, we need to show that if for a given party $s < t - n/7$, then it recovers the correct leaf by the end of the protocol. As we just saw, during the last $n/7$ rounds, that party will set to listen while the other party will set to speak, sending its edge set S_A or S_B. Repeating the above argument and taking into account the additional $n/7$ rounds in which the set of edges (S_A or S_B) already contains the correct path, we find that the correct leaf gets at least $4n/7 - n_0 - B \geq 2n/7$ votes throughout the entire protocol. Any other leaf can get at most $B \leq (2/7 - \varepsilon)n$ votes; thus the output must be the correct leaf. □

While the preceding coding scheme of Algorithm 5 uses alphabet of polynomial size, in [46] an equivalent scheme that uses a constant alphabet size but with a vanishing rate of $O(1/n_0)$ is given. To reduce the alphabet size, each transmission of S_A is replaced with $O(n_0)$ rounds in which the set is encoded using good error-correcting code. This yields a scheme with $O(n_0^2)$ rounds and a constant-size alphabet—thus a rate of $O(1/n_0)$. Using list-decoding techniques similar to the ones discussed in §3, one can obtain an efficient scheme with a constant alphabet and a constant rate [45].

Theorem 4.3 ([45])**.** For any $\varepsilon > 0$, there exists an efficient randomized adaptive coding scheme in the speak-or-listen model that simulates any π_0 in $O_\varepsilon(n_0)$ rounds, uses an alphabet of size $O_\varepsilon(1)$, and succeeds with probability $1 - 2^{-\Omega(n_0)}$, assuming at most a fraction $2/7 - \varepsilon$ of corrupted symbols.

4.2 Adaptive coding schemes: The adaptive-termination and the speak-at-will models

4.2.1 Interactive coding with adaptive length

Consider the following *adaptive-termination model*, in which the party that speaks at round i is predetermined by the round number; however, each party can take, independently at any round of the protocol, an irreversible decision to *terminate*. When a party terminates, it gives an output and stops participating in the protocol. After a party (say, Alice) has terminated (assuming Bob hasn't terminated yet), in each round where Alice is supposed to speak, Bob receives a default symbol $\emptyset$. The channel may still corrupt rounds in which the default symbol $\emptyset$ is sent.

Since the protocol length is not fixed anymore, we need to be more careful in defining the fraction of noise the channel may introduce. The natural way is to define the noise rate *per instance* as the fraction of corruptions out of all the symbols that were sent in that instance. We denote this noise model as *relative noise rate*.

Definition 4.1 (relative noise rate)**.** Given a specific noise pattern, the relative noise rate in a certain instance of the protocol is the fraction of corrupted symbols received by the parties out of the number of symbols the parties sent in that instance.

Adjusting the length of the protocol according to the observed noise leads to a higher noise resilience than the $1/4$ obtained in the nonadaptive case. Specifically, this model, suggested by Agrawal, Gelles, and Sahai [1], allows for interactive coding protocols with a noise resilience of $1/3-\varepsilon$, yet with a vanishing rate.

Theorem 4.4 ([1])**.** Given any function $f(x, y)$ and any $\varepsilon > 0$, there exists an interactive protocol in the adaptive-termination model, that computes f and is resilient to a relative fraction $1/3 - \varepsilon$ of noise.

Proof. The idea of the protocol is the following. Alice and Bob will exchange their inputs x and y encoded via some good error-correcting code (hence, the rate will be vanishing). Yet, the parameters of the encoding (i.e., its length) will depend adaptively on the observed noise. Specifically, Alice starts by sending $\mathsf{ECC}(x)$ using some fixed parameters. Then, Bob estimates the noise in the received word, and chooses his encoding accordingly: the less noise he sees, the more corruption budget still available to the channel, the stronger his encoding would be. The protocol is sketched in Figure 4.1.

Let us analyze the noise the channel makes in case the protocol fails. If Bob aborts before sending y, then clearly the relative noise is $(1-\varepsilon)/2 > 1/3$. Otherwise, there are two cases we need to consider. If Bob decodes $\mathsf{DEC}(\tilde{x}) \neq x$, it must be that the channels has corrupted at least $(1-\varepsilon)N_0 - t$ symbols in Alice's codeword (recall Figure 3.4), and her relative noise in this case will be at least

$$\frac{(1-\varepsilon)N_0 - t}{N_0 + (2N_0 - 4t)} = \frac{N_0 - \varepsilon N_0 - t}{3N_0 - 4t} \geq \frac{1}{3} - O(\varepsilon).$$

In the second case Bob correctly decodes $\mathsf{DEC}(\tilde{x}) = x$, even though the channel has corrupted t symbols of Alice's codeword. The protocol can fail in this case only if Alice incorrectly decodes $\mathsf{DEC}(\tilde{y}) \neq y$. For this to happen, the channel must corrupt $(1-\varepsilon)/2 \cdot (2N_0 - 4t)$ additional symbols of Bob's codeword. This gives a relative noise rate of at least

$$\frac{t + (1-\varepsilon)(2N_0 - 4t)/2}{N_0 + (2N_0 - 4t)} = \frac{N_0 - t - \varepsilon(N_0 - 2t)}{3N_0 - 4t} \geq \frac{1}{3} - O(\varepsilon).$$

0. Alice and Bob decide on a family of codes $\mathsf{ECC}_i : \{0,1\}^{n_0} \to \Sigma^{N_i}$ with $|\Sigma| = O(1)$ and $0 \leq i \leq N_0$. Each ECC_i has a relative distance of $1-\varepsilon$ and length $N_i = N_0 + i$. Furthermore, $\mathsf{ECC}_i(x)$ is a prefix of $\mathsf{ECC}_j(x)$ for any $i < j$. A random linear code will have these properties with high probability.
1. Alice encodes x with via ECC_0 of length $N_0 = O(n_0)$.
2. Bob receives a word $\tilde{x}$ and computes $t = \Delta(\tilde{x}, \mathsf{ENC}(\mathsf{DEC}(\tilde{x})))$.
3. If $t \geq \frac{1}{2}(1-\varepsilon)N_0$, Bob terminates and outputs $\perp$.
4. Otherwise, Bob sends Alice $\mathsf{ECC}(y)$ with $|\mathsf{ECC}(y)| = 2N_0 - 4t$.
5. After Bob finishes sending $\mathsf{ECC}(y)$, he outputs $f(\mathsf{DEC}(\tilde{x}), y)$ and terminates.
6. Alice keeps listening until round $N_0 + 2N_0$ and receives the word $\tilde{y}$ (the length of which is determined by the point after which Alice received only $\emptyset$ symbols)
7. Alice outputs $f(x, \mathsf{DEC}(\tilde{y}))$.

Figure 4.1: Adaptive-length coding scheme with resilience 1/3 in the adaptive-termination model for the identity function $f(x,y) = (x,y)$.

It can also be shown that corrupting the $\emptyset$ symbols in order to make $\tilde{y}$ seem longer or shorter is not helpful for the attack. □

The above protocol only allows the parties to exchange their inputs. Hence, if this protocol is use to simulate an arbitrary protocol it will result with a very bad rate. It is currently open whether a coding scheme with constant rate and noise resilience of $1/3 - \varepsilon$ can be achieved in this model.

On the other hand, 1/2 is an upper bound on the fraction of noise for adaptive protocols that can only change their length.

Theorem 4.5 ([1])**.** No adaptive-length interactive protocol in the adaptive-termination model for the identity function $f(x,y) = (x,y)$ is resilient to a relative noise rate of 1/2 with probability greater than 1/2.

While an upper bound of 1/2 on the noise rate is completely trivial for the nonadaptive case, because there is always a party that speaks at most half of the symbols, this is not the case for adaptive protocols, as the identity of the party that speaks at most half the symbols may change throughout the protocol as a function of the noise. The key idea for proving the bound of 1/2 in the adaptive-termination setting is to corrupt both parties until one of them terminates.

Proof. Assume a protocol π that computes the identity function and tolerates a relative noise fraction of 1/2. Consider an instance of π where Bob holds $y \in \{0,1\}$ and in which we confuse Bob about whether Alice holds a 0 or a 1. That is, regardless of Alice's input, at each round where Alice sends a different symbol in $\pi(0,y)$ and in $\pi(1,y)$, the channel alternates between sending symbols from $\pi(0,0)$ and from $\pi(1,0)$. At the same time, the channel alters Bob's transmission in a similar fashion, alternating between $\pi(x,0)$ and $\pi(x,1)$. Note that this attack is well defined for any input $x, y \in \{0,1\}$.

Without loss of generality, let x, y be the input on which some party P terminates the earliest among all other inputs in $\{0,1\} \times \{0,1\}$. It is clear that P cannot output the correct bit of the

other party with probability greater than 1/2 since P's view is the same whether the other party holds 0 or 1. The channel corrupts about half of all the transmissions until P terminates, so the relative noise is at most 1/2. □

The subtlety that arises in the preceding proof is that we must consider only the party P that terminates first, since the view of the other party may be different if that party terminates after P for one input and not the other; in one of the instances $\emptyset$ is received after P's termination point, which may give the other party an advantage in guessing the input bit of P. A variant of the preceding adaptive-length model that bypasses this subtlety is discussed in [1]. In that variant, the parties are forced to output an invalid output $\perp$ in case they terminate prematurely.

4.2.2 Interactive coding with adaptive order of speaking

The "fully adaptive" *speak-at-will model* combines adaptive termination of the protocol and adaptive choice of whether to speak or not. In contrast to the model of GHS, here the parties always listen; however they can choose whether or not they send a symbol over the channel. In case they choose not to send a symbol, the other side will hear a default $\emptyset$ symbol (unless corrupted by the channel).[1] This allows the parties to terminate the protocol at any desired round by simply giving an output and ignoring symbols received in future rounds.

In this model we again use the notion of relative noise to denote the fraction of corrupted symbols out of all the symbols sent by the parties (i.e., excluding rounds in which a party does not speak).[2] The channel is allowed to corrupt $\emptyset$ and thus may induce a relative noise that exceeds 1. Note that if rounds where no symbol is sent were counted toward the noise budget, then the model would be equivalent to the adaptive-termination model described in §4.2.1.

The ability to remain silent during some of the rounds leads to a new coding approach that has the net effect of doubling the amount relative noise needed to corrupt a single message.

Definition 4.2 (silence encoding)**.** Given a finite message space $\mathcal{M} = [n]$, The k-silence encoding of a message $i \in \mathcal{M}$ is the string

$$k\text{-SE}(i) = \underbrace{\emptyset \cdots \emptyset}_{(i-1)k \text{ times}} \underbrace{11 \cdots 1}_{k \text{ times}} \underbrace{\emptyset \cdots \emptyset}_{(n-i)k \text{ times}} ,$$

where $1 \in \Sigma$ is any non-$\emptyset$ symbol. The decoding of codeword y returns the unique message $m \in \mathcal{M}$ that minimizes $\Delta(y, k\text{-SE(m)})$ or a failure symbol $\perp$ if no such unique message exists.

Lemma 4.6 ([1])**.** For any $k \geq 1$, if k-SE is used to encode a message $m \in \mathcal{M}$, then

1. at least k corruptions are required to make the decoder output $\perp$;

2. at least $k + 1$ corruptions are required to make the decoder output $m' \in \mathcal{M}$, $m' \neq m$.

[1] The motivation behind this model is, for example, an energy-aware channel. Assume the alphabet $\Sigma = \{0, 1, ..., k\}$, encoded via a discrete *amplitude modulation*. That is, in order to send $i \in \Sigma$, the party needs to put i units of energy into the channel. If the party chooses not to spend energy at a given round, the other side hears "silence" that is, the symbol $0 \in \Sigma$.

[2] We stress that this is *not* without loss of generality: choosing not to speak at a given round *does* communicate information to the other side. For this reason, in this model the rate of a protocol should be measured as a function of the round complexity rather than the communication complexity.

Assume $k = 1$ is used to decode symbols. Then, with a single corruption, the receiver will decode a $\perp$ and be aware of the corruption (this is equivalent to an *erasure*). On the other hand, it takes *two* corruptions to convert the symbol into a different one, thus doubling the amount of relative corruption the channel needs to invest in order to make the protocol fail. This idea, along with the protocol of Figure 4.1, or along with Algorithm 1 gives the following.

Theorem 4.7 ([1]). Given any function $f(x, y)$ and any $\varepsilon > 0$, there exists an interactive protocol in the speak-at-will model, that computes f and is resilient to a relative fraction $2/3 - \varepsilon$ of noise. In the worst case, the protocol has a vanishing rate (with respect to the shortest noiseless protocol for $f(x, y)$).

Theorem 4.8 ([1]). For any $\varepsilon > 0$ there exists an interactive coding scheme that simulates any protocol π_0 in $O_\varepsilon(n_0)$ rounds, uses a binary alphabet and is resilient to a relative fraction $1/2 - \varepsilon$ of noise.

It is rather straightforward that 1 is a trivial upper bound on the relative noise rate of this model; with such a high corruption budget the channel can erase the entire communication, that is, turn any symbol into $\emptyset$.

Theorem 4.9 ([1]). No interactive protocol in the speak-at-will model can compute a nonconstant function $f(x, y)$ and is resilient to a relative noise rate of 1 with probability greater than $1/2$.

Open Questions for Section 4

1. Is it possible to achieve coding schemes with varying length (in the adaptive-termination model) with resilience at least $1/3 - \varepsilon$ and a constant rate?
2. What is the maximal noise resilience for protocols with varying length? Specifically, is it possible to achieve a coding scheme with resilience $1/2 - \varepsilon$ (even one with a vanishing rate)?
3. What is the maximal noise resilience of an adaptive constant-rate coding scheme with both varying length and adaptive order of speaking?

5

Communication-Efficient Coding Schemes

So far we have concentrated on coding schemes with a constant (nonvanishing) rate; however we didn't care much about the actual constant. In this section we ask how large (how close to 1) the rate of interactive coding schemes can get, as a function of the noise. Throughout this section, we will assume the noise fraction ε is small, and analyze the asymptotic behavior of the rate as ε tends to 0.

The above question is well studied for the unidirectional setting, where a sender holds a message from some message space $\mathcal{M}$ and the goal is to communicate this message to a receiver utilizing a noisy channel. The pioneering work of Shannon [80] proves that asymptotically (i.e., for a large-enough message space), a rate slightly below $1 - H(\varepsilon)$ is enough for coding over a BSC_ε that succeeds with exponentially high probability (in the length of the message). $H(\cdot)$ is the binary entropy function $H(x) = x \log \frac{1}{x} + (1-x) \log \frac{1}{1-x}$. A similar result (up to the constants) holds for the case where up to a fraction ε of the bits may be corrupted [53]; a coding scheme with a rate of $1 - O(H(\varepsilon))$ is still achievable.

We now extend the discussion to the interactive case. In order to obtain *communication-efficient* coding schemes with a rate that approaches 1, it seems that there is no other choice but simulating the (noiseless) protocol as is while trying to figure out whether errors have happened, and then quickly correcting any errors that may have happened. That is, the rewind-if-error paradigm (Figure 2.3) is the prominent one for coding schemes with a rate that approaches 1 as the noise approaches 0.

Unlike the unidirectional setting, when an error happens in an interactive protocol the rest of the information (which is based on the incorrect history) is useless. Thus, in order to obtain communication-efficient coding schemes, it is important to be able to find and correct errors very close to the time which they occurred, which implies one needs to check the correctness of the simulation very frequently.

The above realization suggests that the maximal obtainable rate is $1 - O(\sqrt{\varepsilon})$. Assume we run k bits of the protocol and then perform some checks to see if errors have happened, say, by communicating c bits. Then, there are two contradictory goals in setting k: On one hand, we want k to be large, since we add c bits every k bits of the protocol, that is, we lose a factor of c/k in the

rate. On the other hand, as k gets larger, so does the probability of an error during this chunk. Such an error will cause the simulation to rewind at least one chunk, increasing the simulation by at least k additional bits.

Assuming a channel with random noise BSC_ε, the preceding reasoning suggests that the coding scheme needs to communicate at least

$$n_0 + c \cdot \frac{n_0}{k} + (k+c) \cdot \left(\frac{n_0}{k} \cdot \varepsilon(k+c) \right)$$

bits in expectation: n_0 bits for the (noiseless) simulation of π_0, c checkup bits for every k-bit chunk gives the second term, and finally, for each one of the expected $\frac{n_0}{k} \cdot \varepsilon(k+c)$ errors, we need to re-simulate the noisy chunk which costs another $c+k$ bits and gives the third term. The minimum is obtained when choosing $k \approx \sqrt{c/\varepsilon}$, which explains the square-root factor loss in the rate.

A naive attempt to convert Algorithm 2 into an efficient-communication one involves reducing the block size (k) from $\log n_0$ to $O(1/\sqrt{\varepsilon})$. Similarly, the output of the hash function in use must be restricted to a small constant (e.g., $O(\log 1/\varepsilon)$, for a total rate of $1 - \tilde{O}(\sqrt{\varepsilon \log 1/\varepsilon})$). However, this naive conversion faces a difficulty when the parties need to communicate the length $|T|$ of their currently simulated transcript. Communicating this value directly takes $\log n_0$ bits, which is too high. On the other hand, communicating only a small hash of this value reveals only whether or not they have the same length, but it doesn't reveal which party is ahead of another; thus they cannot synchronize back to a point where they agree.

There are two approaches to solve a possible mismatch in the length of T without communicating its length: (1) With fixed point of return, the parties predetermine certain rounds of the simulation to perform a consistency check. If the check fails, the simulation is rewound to a fixed and predetermined point. (2) In a dynamic point of return, the parties dynamically find a common prefix of T where they both agree. To this end, they start "going back" in T in an increasing magnitude until they find a point of agreement.

In the following we explore two schemes that take these two paths. The first scheme, by Kol and Raz [59], uses fixed return points, while the second scheme, by Haeupler [51], takes the second approach and dynamically coordinates rendezvous points.

5.1 Rate $1 - O(\sqrt{\varepsilon \log 1/\varepsilon})$ for random noise

The first communication-efficient protocol, suggested by Kol and Raz [59], achieves a rate of $1 - O(\sqrt{\varepsilon \log 1/\varepsilon}) = 1 - O(\sqrt{H(\varepsilon)})$ over a BSC_ε. The protocol takes a recursive rewind-if-error approach, which can be seen as an optimized version of the checking steps of the scheme in [74]: after every chunk of data the parties exchange some bits to verify the correctness of the currently simulated transcript. However, as the transcript gets longer and longer, so does the number of checking bits communicated to verify the transcript and so does the number of steps we allow the simulation to rewind. The checking steps are built in recursive levels that can be thought of as a k-ary tree: each chunk of the noiseless protocol is a leaf of the tree (level 0), and each node above it (level ℓ) is a checking point that verifies all the levels below it and rewinds the protocol 2^ℓ steps back in case of inconsistency. The coding scheme is described in Figure 5.1.

Lemma 5.1 ([59]). The KR protocol (Figure 5.1) has a rate of $1 - O(\sqrt{\varepsilon \log 1/\varepsilon})$.

Protocol A_1 (assuming currently simulated transcript T_{prev}):

1. Run π_0 for k steps, assuming the simulated transcript so far is T_{prev}. Let T be the new k simulated bits.
2. Let $\mathcal{H} = \{h : \{0,1\}^k \to \{0,1\}\}$ be a family of $101 \log k$ hash functions, shared between the parties (e.g., via a shared random string).
3. For any $h \in \mathcal{H}$ send $\mathsf{ECC}(h(T))$, concretely, use a repetition code of length 101 bits.
4. If all the (decodings of the) received hash values are consistent with T, output $T_{prev} \circ T$. Otherwise, output T_{prev}.

Protocol $A_{\ell+1}$ (assuming currently simulated transcript T_{prev}):

1. $T \leftarrow T_{prev}$.
2. Repeat k times: $T \leftarrow A_\ell$ (assuming the currently simulated transcript is T).
3. Let $\mathcal{H} = \{h : \{0,1\}^{k^{\ell+1}} \to \{0,1\}\}$ be a family of $O(101^{\ell+1} \log k)$ hash functions.
4. For any $h \in \mathcal{H}$ send $\mathsf{ECC}(h(T))$, concretely, use repetition code of length $101^{\ell+1}$.
5. If all the (decodings of the) received hash values are consistent with T, output T. Otherwise, output T_{prev}.

The KR simulation:
Given a protocol π_0 to simulate and a noise parameter ε, set k so that $\varepsilon = \frac{\log k}{k^2}$, and let $s = c \log n_0$ for some small-enough c. Run A_s sequentially for $\beta = \frac{n_0}{k^s}(1 + O(\frac{\log k}{k}))$ times.

Figure 5.1: The Kol-Raz coding scheme [59].

Proof. The subprotocol A_ℓ communicates $k \cdot \mathsf{CC}(A_{\ell-1}) + 2 \cdot 101^{2\ell} \log k$ bits. By opening the recursion, we can bound this amount by $k^\ell(1 + O(\frac{\log k}{k}))$. Therefore, the KR scheme communicates a total of

$$\begin{aligned}\mathsf{CC}(A_s) \cdot \frac{n_0}{k^s}\left(1 + O\left(\frac{\log k}{k}\right)\right) &\le k^s\left(1 + O\left(\frac{\log k}{k}\right)\right) \cdot \frac{n_0}{k^s}\left(1 + O\left(\frac{\log k}{k}\right)\right) \\ &= n_0\left(1 + O\left(\frac{\log k}{k}\right)\right)\end{aligned}$$

bits. Recall that $\varepsilon = (\log k)/k^2$, then the above equals

$$n_0\left(1 + O\left(\sqrt{\varepsilon \log k}\right)\right) = n_0\left(1 + O\left(\sqrt{\varepsilon \log 1/\varepsilon}\right)\right). \qquad \square$$

Theorem 5.2 ([59]). The KR scheme simulates π_0 over a BSC_ε with probability $1 - 2^{-n_0^{\Omega_\varepsilon(1)}}$.

Proof. For each subprotocol A_ℓ, define the following two measures:

1. Disagreement probability: the probability that, given both parties assumed the simulated transcript so far was T_{prev}, they end up with different outputs.
$$\mathsf{dis}_\ell = \Pr[A_\ell(T_{prev})^{\text{Alice}} \neq A_\ell(T_{prev})^{\text{Bob}}].$$
2. Progress: the expected increase in the length of T, given that both parties started with the same T_{prev}.
$$\mathsf{prog}_\ell = \mathbb{E}\left[|A_\ell(T_{prev})| - |T_{prev}|\right].$$

For the base case, A_1, the parties would disagree on the output in one of two cases: (1) an error occurred in the simulation, but all the hash values came out the same (i.e., there was a hash collision between the simulated transcript T computed at Alice's and Bob's sides); or (2) the transcripts in both sides are the same, but at least one of the received hashes is inconsistent (due to noise in transmitting this hash value).

The probability of the first case is at most $2^{-101 \log k}$: we can choose the hash functions so that a collision will have a probability 1/2 (e.g., by taking the binary inner product hash $h_a(x) = \langle x, a \rangle$). The probability of the second case is the probability that the noise flips more than 51 bits of the encoding of a single hash, which is bounded by $2 \cdot 101 \log k \cdot 2^{101} \cdot \varepsilon^{51} \leq \varepsilon^{20}$ (we can change the constants to force this inequality in case ε is not small enough). Thus, we can safely assume $\mathsf{dis}_1 < k^{-20}$.

As for the progress of A_1, note that the simulation progresses by k steps unless there was at least one bit flip in the entire communication of A_1, which happens with probability at most $\varepsilon \cdot (k + 2 \cdot 101 \cdot 101 \log k) \leq O(\frac{\log k}{k})$. Thus, $\mathsf{prog}_1 \geq k(1 - O(\frac{\log k}{k}))$.

Repeating a similar argument for the ℓ-th level of the recursion A_ℓ, one gets,

$$\begin{aligned}
\mathsf{dis}_\ell &\leq k^{-20^\ell}, \\
\mathsf{prog}_\ell &\geq k \cdot \mathsf{prog}_{\ell-1}(1 - k^{-10^{\ell-1}}) \geq k^\ell \left(1 - O\left(\frac{\log k}{k}\right)\right).
\end{aligned}$$

And for the entire KR protocol that runs A_s sequentially for $\beta = \frac{n_0}{k^s}(1 + O(\frac{\log k}{k}))$ times, we have a disagreement probability of

$$\begin{aligned}
\mathsf{dis}_{KR} &\leq \beta \cdot \mathsf{dis}_s \\
&\leq n_0 \cdot \mathsf{dis}_s \\
&\leq n_0 k^{-20^{c \log n_0}} \\
&\leq n_0 k^{-n_0^{\Omega(c)}} = 2^{-n_0^{\Omega_\varepsilon(1)}},
\end{aligned}$$

and the expected progress is given by the progress of all the A_s instances, assuming they all ended without any disagreement (which happens w.p. at least $(1 - \beta \mathsf{dis}_s)$,

$$\begin{aligned}
\mathsf{prog}_{KR} &\geq \beta \mathsf{prog}_s \cdot (1 - \beta \mathsf{dis}_s) \\
&\geq \frac{n_0}{k^s}\left(1 + O\left(\frac{\log k}{k}\right)\right) \cdot \left(1 - 2^{-n_0^{\Omega_\varepsilon(1)}}\right) \cdot k^s \left(1 - O\left(\frac{\log k}{k}\right)\right) \\
&\geq n_0 \left(1 + O\left(\frac{\log k}{k}\right)\right),
\end{aligned}$$

where the last transitions can hold with an appropriate choice of constants. This implies that, except with probability $2^{-n_0^{\Omega_\varepsilon(1)}}$ for some small constant c, the parties agree on the output transcript, and that the simulated transcript has an expected length greater than $n_0(1 + O(\sqrt{\varepsilon}))$. Therefore, via a tail bound on the length of the transcript, except with probability of $2^{-n_0^{\Omega_\varepsilon(1)}}$, the parties will have successfully simulated at least n_0 steps of π_0. □

5.2 Rate $1 - O(\sqrt{\varepsilon})$ for random noise

A coding scheme that achieves a rate of $1 - O(\sqrt{\varepsilon})$ was given by Haeupler [51].

The scheme, depicted in Algorithm 6, generally follows the rewind-if-error paradigm using chunks of size $k \approx O(1/\sqrt{\varepsilon})$ and hash values of size $c \approx O(1)$. In addition to exchanging hash values of the currently simulated transcript, the parties also exchange hash values of two additional *meeting points* in the history to which the simulation may revert (this is an optimized version of the meeting-points mechanism in [74]). This mechanism allows the parties to coordinate rewinding the protocol in a synchronized way by communicating hashes of $O(1)$ bits, maintaining the communication efficiency of the scheme.

To see the need for such a mechanism, recall the simple rewind-if-error scheme of Algorithm 2. When the two parties find that their transcripts are inconsistent (i.e., the hashes mismatch), they need to rewind. Note that they must rewind in a coordinated way: if one party is ahead of the other, only that party must rewind or otherwise, if both rewind, their transcripts would remain inconsistent until they rewind to the very first round. Indeed, in Step 9 of Algorithm 2, the parties identify the party that is ahead and only this party rewinds (both parties rewind if both transcripts are of equal length). This step is possible since Algorithm 2 uses a large alphabet size that allows communicating the current length of the transcript. However, when the alphabet size is small, we cannot communicate this information, and a more sophisticate method must be used in order to determine which party needs to rewind, and by how many rounds.

It is crucial that a small amount of errors would force the parties to rewind only a small number of rounds. Put differently, if the parties agree to rewind a large number of rounds, then it must be the case that many transmissions were corrupted by noise. The meeting points mechanism dynamically sets the number of rounds to rewind as a function of the noise, so that a small amount of noise will not be able to cause the simulation to rewind too many rounds.

More specifically, the mechanics of the meeting points is as follows. When an inconsistency is found (hashes mismatch), the parties try to agree on a point that is at most $2 \times 2^\ell$ chunks away from their current position (starting with $\ell = 1$). To this end, they define as meeting points the two longest transcript prefixes whose length is an integral multiply of 2^ℓ. For example, if the length (in chunks) of the current transcript is $t2^\ell + r$ with $0 \leq r < 2^\ell$, then the two meeting points will be the prefix of length $t2^\ell$ chunks and the one of length $(t-1)2^\ell$ chunks. Whenever the parties compare hashes of the current simulated transcript, they also compare hashes of the transcript prefixes corresponding to each of the two meeting points. Each time these hashes are equal for some meeting point, that meeting point gains more confidence. Eventually, after 2^ℓ rounds the parties make a decision whether or not to rewind to a meeting point: the parties revert to a meeting point that gained confidence greater than about $2^\ell/2$, if such exists. Otherwise, ℓ is increased by 1, and the parties repeat this procedure, now with new meeting points located further back in the history. To be certain that the parties are synchronized with respect to the meeting points' mechanism, they also exchange hashes of the meta data regarding their ℓ and meeting-point position. If during the 2^ℓ rounds, more than half of the metadata hashes are inconsistent, the parties reset $\ell = 1$ and try again. This prevents the adversarial channel from interfering with the meeting points mechanism itself.

Remark 5.1. In the coding scheme, described in Algorithm 6, we assume the parties pre-share a random string of length $O(n_0^2)$; this randomness is used to seed hash functions. However, this assumption is not needed, and in fact the parties can exchange randomness while keeping the communication small. This is done by exchanging about εn_0 random bits, encoded via a good error-correcting code (say, with distance $4\varepsilon n_0$; this can be done by communicating $O(\varepsilon n)$ bits without

Algorithm 6 Haeupler's coding scheme with rate $1 - O(\sqrt{\varepsilon})$ [51]

Input: a protocol π_0 of length $n_0 = |\pi_0|$ and an input x.

Let $c = O(1)$ (the hash size), $k = O(\sqrt{c/\varepsilon})$ (the size of each chunk).
Assume R is a shared random string of length $O({n_0}^2)$.
Let $N = n_0/k + 65n_0\varepsilon$ and set $n = Nk = n_0(1 + O(\sqrt{\varepsilon}))$.
Let $\mathcal{H} = \{h_s : \{0,1\}^n \to \{0,1\}^c\}$ be a family of hash functions with seed size $|s| = O(c \cdot (n + \log n))$ and input size long enough to hash a complete transcript of the scheme below. The i-th bit of $h_s(x)$ is defined as the inner product $\langle x \circ |x|, s_{(i-1)(n+\log n)+1} \cdots s_{i\cdot(n+\log n)} \rangle$.

$T \leftarrow \emptyset$; $gap \leftarrow 0$.
for $i = 1$ to N **do**
  $s \leftarrow$ new seed from R
  compute new estimated gap between the transcripts, and the meeting points:
    $gap \leftarrow gap + 1$
    $\ell \leftarrow \lfloor \log gap \rfloor$
    write $|T|/k = m \cdot 2^\ell + rem$, with $m \in \mathbb{N}$ and $rem \in [0, 2^\ell - 1]$, then
    $\mathsf{mp1} = (m \cdot 2^\ell)k$, $\mathsf{mp2} = \max(0, (m-1)2^\ell)k$
  let $(H_{gap}, H_T, H_{mp1}, H_{mp2}) \leftarrow (hash_s(gap), hash_s(T), hash_s(T[1..\mathsf{mp1}]), hash_s(T[1..\mathsf{mp2}]))$
  send $(H_{gap}, H_T, H_{mp1}, H_{mp2})$
  respectively, receive $(H'_{gap}, H'_T, H'_{mp1}, H'_{mp2})$

  if $H_{gap} \neq H'_{gap}$ **then** ▷ the parties don't agree on the value of gap (i.e., on 2^ℓ)
    $gap\text{-}err \leftarrow gap\text{-}err + 1$
  else ▷ maintain the meeting point that is more consistent
    if $H_{mp1} = H'_{mp1}$ or $H_{mp1} = H'_{mp2}$ **then**
      $\mathsf{mp1}.vote \leftarrow \mathsf{mp1}.vote + 1$
    if $H_{mp2} = H'_{mp1}$ or $H_{mp2} = H'_{mp2}$ **then**
      $\mathsf{mp2}.vote \leftarrow \mathsf{mp2}.vote + 1$

  Simulate next chunk (if transcripts seem consistent):
  if $(H_{gap}, H_T, H_{mp1}, H_{mp2}) = (H'_{gap}, H'_T, H'_{mp1}, H'_{mp2})$ and $gap = 1$ **then**
    run k more steps of π_0 assuming the simulated transcript so far is T
    $gap \leftarrow 0$, $\mathsf{mp1}.vote \leftarrow 0$, $\mathsf{mp2}.vote \leftarrow 0$
  else
    perform k dummy simulation rounds (ignore incoming communication)

  Rewind mechanism:
  if $2 \cdot gap\text{-}err \geq gap$ **then** ▷ meta data cannot be trusted
    $gap\text{-}err, gap \leftarrow 0$
  else if $gap = 2^\ell$ **then** ▷ Simulation reached a checkpoint
    ▷ rewind if some meeting point gained enough trust.
    if $\mathsf{mp1}.vote \geq 0.4 \cdot 2^\ell$ **then**
      rewind back to $\mathsf{mp1}$; $gap \leftarrow 0$
    else if $\mathsf{mp2}.vote \geq 0.4 \cdot 2^\ell$ **then**
      rewind back to $\mathsf{mp2}$; $gap \leftarrow 0$
    $\mathsf{mp1}.vote, \mathsf{mp2}.vote \leftarrow 0$ ▷ reset counters for the next gap level
output T (a prefix of length n_0)

affecting the rate of the interactive coding scheme). Then, this short random string is expanded into a string of length $O(n_0^2)$, which is very close to uniform—for example, via small-biased probability spaces [67, 3]. [The above expansion of randomness used as a seed for hash functions can be seen as a simple application of Theorem 2.8.] To make explanation simpler, we will assume from this point on that the parties share a uniform string of the necessary length.

Lemma 5.3 ([51])**.** Algorithm 6 has a rate of $1 - O(\sqrt{\varepsilon})$.

Proof. The scheme runs for $N = n_0/k + 65n_0\varepsilon$ rounds, in each of which the parties communicate $8c + k$ bits. The total communication is thus

$$\begin{aligned} N(8c+k) &= n_0 \cdot \left(1 + \frac{8c}{k}\right) + \varepsilon(65n_0(8c+k)) \\ &= n_0 \left(1 + O\left(\sqrt{\varepsilon}\right) + O(\varepsilon)\right) \\ &= n_0 \left(1 + O\left(\sqrt{\varepsilon}\right)\right), \end{aligned}$$

which proves the claim (recall that $\varepsilon < 1$; therefore, $\varepsilon \leq \sqrt{\varepsilon}$). □

Theorem 5.4 ([51])**.** Algorithm 6 simulates any π_0 over a BSC_ε with probability $1 - 2^{-\Omega_\varepsilon(n_0)}$.

Since the complete proof is very technical, we only give here some intuition and refer the reader to [51] for full details.

Proof intuition. The proof basically goes along the same lines of proving the correctness of Algorithm 2; that is, we define a potential function that measures the length of the correct prefix of the simulated transcript. It is easy to see that as long the two sides are fully synchronized, $gap = 1$, and no error has happened, then the parties simulate the k rounds correctly and increase the potential. However, errors may decrease the potential either by corrupting the simulated transcript, or by causing the parties to get out of sync (i.e., by corrupting the metadata H_{gap}, etc.). Additionally, an event of hash collision may prevent the parties from learning they are not synchronized and delay the simulation even further.

Very intuitively, in the worst case, a burst of errors may eventually cause the simulation to rewind up to $2 \cdot 2^\ell$ chunks (e.g., by causing enough errors to make $gap = 2^\ell$); however, this can happen only if the rewind mechanism at the round where $gap = 2^{\ell-1}$ did not make the parties go back, which implies $\approx 2^\ell/4$ errors (or hash collisions) must have happened during the last 2^ℓ iterations. It follows that, for any constant c, a total of $c \cdot \varepsilon n_0$ errors can rewind the simulation by at most $O(c\varepsilon n_0)$ chunks of size k. Note that for $c > 1$, the probability that over a BSC_ε there are more than $c\varepsilon n_0$ is at most $2^{-\Omega_\varepsilon(n_0)}$. Additionally, it is shown that the number of hash collisions that happen in iterations where the parties are *unsynchronized*[1] is at most $O(\varepsilon n)$ with high probability. Hence, running the coding scheme for $n_0/k + O(\varepsilon n_0)$ iterations with a large-enough constant suffices to correctly simulating π_0. □

If the coding scheme is allowed to be adaptive, then Algorithm 6 obtains the same rate for any π_0, even non-alternating one. We note that the adaptivity here is needed: assume that π_0 is not alternating, but instead the order of speaking is such that Alice speaks for k rounds and then

[1] Recall that a hash collision happens when $h(x) = h(y)$ but $x \neq y$. Thus, iterations in which the parties are synchronized (i.e., where $x = y$) should not be counted toward hash collisions!

Bob talks for k rounds. If the coding scheme is not adaptive, then after each rewind the order of speaking in the coding scheme may mismatch the one of π_0, which causes a loss in the rate.

Using similar methods to Algorithm 6, Haeupler [51] showed a coding scheme that tolerates up to a fraction ε of *adversarial* noise and has a rate $1 - O(\sqrt{\varepsilon \log\log 1/\varepsilon})$.

Theorem 5.5 ([51])**.** For any small-enough ε, there exists an efficient interactive coding scheme that simulates π_0 with probability $1 - 2^{-\Omega_\varepsilon(n_0)}$ in the presence of up to a fraction ε of adversarial noise. The scheme has a rate of $1 - O(\sqrt{\varepsilon \log\log 1/\varepsilon})$.

5.3 Rate upper bounds and the order of speaking

The question that emerges from the above discussion is, *"what is the capacity of interactive communication"*, that is, what is the maximal rate that coding protocols can achieve. Formally, we can define the capacity of interactive coding in the following manner:

Definition 5.1. The *capacity* of interactive protocols over a channel Ch is

$$C(\mathsf{Ch}) = \liminf_{n_0 \to \infty} \min_{\substack{f \text{ s.t.} \\ \mathsf{CC}(f) = n_0}} \frac{n_0}{\mathsf{CC}_{\mathsf{Ch}}(f)} \ ,$$

where $\mathsf{CC}_{\mathsf{Ch}}(f) = \min_\pi \mathsf{CC}_{\mathsf{Ch}}(\pi)$ over all the protocols π that compute f over Ch with a vanishing error probability over the parties' coin- flips (and the noise, in case it is random).

Note that we define the capacity using limit inferior, since it is not known whether the best achievable rate, as $n_0 \to \infty$, converges to a limit.

It is very interesting to compare this quantity to the capacity of a coding over a unidirectional noisy channel, known to be precisely $1 - H(\varepsilon)$ for a BSC_ε and $1 - \Theta(H(\varepsilon))$ for a fraction ε of adversarial noise (with a constant less than 2). In the interactive case, as suggested before, the rate is likely to be closer to $1 - O(\sqrt{\varepsilon})$; hence there is a gap between the unidirectional and interactive settings; see Figure 5.2.

A first upper bound on the rate of coding schemes in the interactive setting was proven by Kol and Raz [59], showing a specific noiseless protocol whose (nonadaptive) simulation over a BSC_ε has a rate at most $1 - \Omega(\sqrt{H(\varepsilon)})$.

Theorem 5.6 ([59])**.** For any $d, k \in \mathbb{N}$, there exists a (non-alternating) protocol π_0 that computes a specific distributed function by communicating $\mathsf{CC}(\pi_0) = d \cdot k$ bits over a noiseless channel, such that any coding scheme π with a predetermined order of speaking solves the same task over a BSC_ε with $\varepsilon = O(\frac{\log k}{k^2})$ and success probability δ, communicates at least

$$\mathsf{CC}(\pi) \geq d \cdot (k + \Omega(\log k))(1 - 2\delta) - O(k)$$

bits, in expectation.

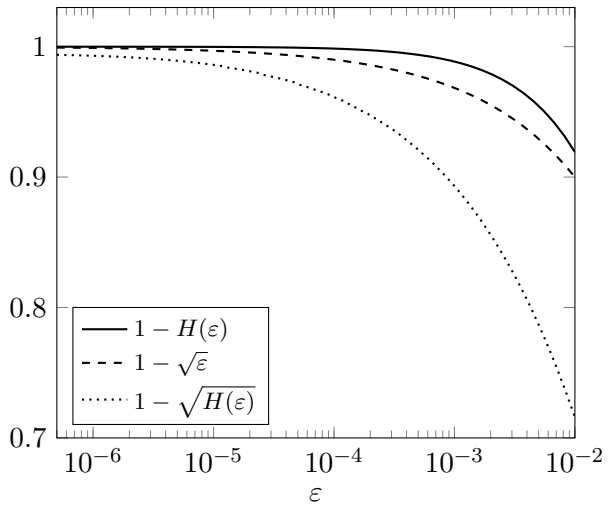

Figure 5.2: The rate gap between the interactive and the noninteractive case for small ε (omitting the constant in the leading term)

Assuming large-enough d, k, the theorem implies that any coding scheme for the task mentioned in the theorem has a rate r bounded above by

$$\begin{aligned} r &\leq \frac{dk}{d \cdot (k + \Omega(\log k))(1 - 2\delta) - O(k)} \\ &\approx \frac{1}{1 + \Omega(\log k)/k} \\ &= 1 - \Omega\left(\frac{\log k}{k}\right) \\ &= 1 - \Omega\left(\sqrt{\varepsilon \log 1/\varepsilon}\right) = 1 - \Omega\left(\sqrt{H(\varepsilon)}\right). \end{aligned}$$

Based on a superficial look, the above theorem seems to conflict with the existence of Algorithm 6, which achieves a rate of $1 - O(\sqrt{\varepsilon}) > 1 - \Omega(\sqrt{H(\varepsilon)})$. Yet, the two differ in the setting they assume. Specifically, they differ in the order in which the parties speak in the noiseless protocol: the upper bound of Theorem 5.6 assumes a particular order of speaking in π_0, namely, each party talks for k consecutive rounds, where k is determined as a function of ε. In Algorithm 6, on the other hand, it is assumed that the simulated protocol π_0 is alternating.

It follows that the order of speaking in π_0 greatly affects the rate of any (nonadaptive) coding scheme. Note that any protocol π_0 can be turned into an equivalent *alternating protocol*; however, this may cause the increase of the communication by a factor of up to 2. In other words, this limits the rate by at most half, in the worst case.

The result of Kol and Raz (Theorem 5.6) provides a protocol π_0 with a very certain order of speaking and shows lower bound on the rate of any coding scheme with a predefined order of speaking for that π_0 (even if the order of the simulation is set according to π_0). Haeupler [51] conjectured that, unless the simulation is adaptive, there exist noiseless protocols for which the rate of any simulation with a fixed order of speaking is bounded away from 1.

Conjecture 5.1 ([51])**.** Given π_0 whose order of speaking is sufficiently irregular (e.g., pseudo-random), any fixed-order (nonadaptive) simulation of π_0 has a rate which is bounded away from 1.

Overcoming this conjectured impossibility can be done by using *adaptive* coding schemes—it is reasonable that Algorithm 6 is capable of simulating any adaptive protocols with rate $1 - O(\sqrt{\varepsilon})$ (say, in a variant of the speak-or-listen model with a random noise). Upper bounds on the rates of adaptive coding schemes (either in the speak-or-listen or the speak-at-will model) are still open.

Open Questions for Section 5

1. What is the maximal rate of alternating coding schemes, assuming an alternating π_0? That is, prove or refute the conjecture [51] that the best coding scheme has rate $1 - \Theta(\sqrt{\varepsilon})$.
2. Prove (or refute) Conjecture 5.1: show a π_0 for which no nonadaptive simulation with a rate of $1 - o(1)$ exists.
3. What is the capacity of *adaptive* interactive protocols (that is, the asymptotical maximal rate obtainable by adaptive coding schemes), either in the speak-or-listen or the speak-at-will model?
4. How is the rate of interactive coding schemes affected by various setting assumptions (e.g., a shared randomness, a reduced success probability, etc.)?

6

Coding Schemes over Different Noisy Channels

In this section we discuss coding schemes over channels and noise models that are different from the "standard" BSC_ε and adversarial noise discussed before. Both the noise resilience and the rate of the coding scheme greatly depend on the specific noise model. The analysis of different relevant noise models could possibly lead to coding schemes that perform better in many common situations and settings.

First we discuss a *weaker* type of noise, namely, channels with noiseless feedback and erasure channels. These models are weaker than the standard noise model because here at least one of the parties is aware of the error: when noiseless feedback channels are present, the sender learns the symbol received at the other side (and thus can compare it to the sent symbol); in the case of erasure noise, the receiver learns that a certain symbol was erased. One can also consider erasure channels with noiseless feedback; however in this type of channel both parties are aware of an error and can simply repeat the transmission until the symbol is received correctly at the other side [76].

Next we discuss a *stronger* type of noise, namely, a channel that can insert and delete symbols. This type of noise causes the parties to believe they are at different steps of the coding scheme.

Lastly we briefly mention a coding scheme that assumes quantum channels, where the parties exchange quantum systems rather than classical bits.

6.1 Channels with noiseless feedback

In the case of channels with feedback, we assume that in addition to the noisy ("main") channel, the parties share another *noiseless* ("feedback") channel. However, the additional channel can be used only in a very restricted manner: when a symbol σ is communicated through the noisy channel and a symbol σ' (possibly corrupted) is received, the symbol σ' is then automatically sent over the feedback channel back to the sender. This way, the sender can learn whether his symbol was correctly received at the other side and, if not, what symbol was received. As before, the main channel can be a BSC_ε or a channel with a fraction ε of adversarial noise.

We begin by discussing the noise resilience of coding schemes in this model, and show optimal schemes with constant rate over channels with binary and larger alphabets. Then, we discuss

communication efficient coding schemes and the maximal rate achievable in this model.

6.1.1 Noise resilience

Nonadaptive coding schemes with fixed order of speaking

When a noiseless feedback is present, the task of identifying whether the simulation so far is correct or not becomes trivial; at least one side is aware of previous errors. That party can send a special symbol to instruct the other side to rewind the simulation until all errors are corrected. Following this observation, Efremenko, Gelles, and Haeupler [27, 28] designed a very simple and efficient rewind-if-error coding scheme that is resilient to a fraction $1/4 - \varepsilon$ of noise, which is described in Algorithm 7.

The coding scheme uses a ternary alphabet $\{0, 1, \leftarrow\}$, where $0, 1$ are used to denote information bits and '$\leftarrow$' is a special symbol that upon reception, instructs both parties to rewind the simulation of π_0 by 3 rounds. The parties exchange symbols in alternating rounds, and maintain a string T describing the simulated transcript so far. When an information bit $b \in \{0, 1\}$ is received, this bit is added to T (in both sides). When '$\leftarrow$' is received, both sides erase the last three bits of T. Note that due to the feedback, both parties know which symbols were *received*, and so both parties always hold the same T (which may be correct or may contain errors).

The coding scheme assumes a *fixed* order of speaking, specifically, the parties send symbols in an alternating manner. This must hold also when a $\leftarrow$ symbol is received. For this reason, the parties rewind 3 rounds (rather than, say, only 2), and keep the order of speaking alternating; see Figure 6.1. This has the side effect of reducing the noise resilience (see below for scheme with nonfixed order of speaking).

Theorem 6.1 ([28])**.** For any $\varepsilon > 0$, Algorithm 7 communicates $O_\varepsilon(n_0)$ bits and is resilient to a fraction $1/4 - \varepsilon$ of corruptions.

Proof. It is clear that the algorithm communicates $O(n) = O_\varepsilon(n_0)$ bits, and we need to show only that it tolerates noise rate of $1/4 - \varepsilon$.

The main observation is that each error causes the simulation to "stall" for exactly 4 rounds. Consider an error at round i, say changing the value of a communicated information bit. If during the next 3 rounds there are no errors, then at round $i + 2$ the party will send $\leftarrow$, and at the end of round $i + 3$ both parties will hold the same T as they had at the end of round $i - 1$. See Figure 6.1 for an illustration. The same happens if a bit b is turned by the noise into a $\leftarrow$ symbol, or vice versa.

Furthermore, if an error happens at round i and an additional error happens at round i', where $i < i' \leq i + 3$, then it still takes at most 3 noiseless rounds to get the parties back to the state they were at the end of round $i' - 1$; thus, after at most 6 noiseless rounds, they are back to the state they were at the end of round $i - 1$. That is, errors just linearly accumulate.

The preceding analysis suggests that $n(1/4 - \varepsilon)$ errors are fully corrected after $3n(1/4 - \varepsilon)$ noiseless rounds. In the other $n - 3n(1/4 - \varepsilon) - n(1/4 - \varepsilon) = 4\varepsilon n = n_0$ rounds, the simulation progresses without disruption and extends the simulated transcript T by a single bit each time. Therefore, at the end of the simulation, the correct prefix of T is at least n_0 bits long. □

It is possible to convert Algorithm 7 to use the binary alphabet, by encoding the $\leftarrow$ symbol by two bits, say 00, and ensuring that the noiseless protocol π_0 never sends two consecutive zeros (e.g.,

Algorithm 7 A coding scheme over channels with feedback with ternary alphabet [28]

Input: a binary alternating protocol π_0 and an input value x. A noise parameter $1/4 - \varepsilon$.

Initialize $T \leftarrow \emptyset$; $T^F \leftarrow \emptyset$.
Split T into two substrings corresponding to alternating indices: T^S are the sent characters, and T^R the received characters. Let T^F be the characters received by the other side (as learned via the feedback channel).

1: **for** $i = 1$ to $n = \lceil n_0/4\varepsilon \rceil$ **do**
2: **if** $T^F = T^S$ **then** ▷ no errors in T are known
3: $T \leftarrow T \circ \pi(x \mid T)$ ▷ run one step of π, given the transcript so far is T
4: $T^F \leftarrow T^F \circ \langle$symbol recorded at the other side$\rangle$ ▷ only if sender
5: **else** ▷ T contains errors, send a rewind request
6: if sender:
7: send a '$\leftarrow$' symbol
8: $T \leftarrow T \circ$ '$\leftarrow$'
9: $T^F \leftarrow T^F \circ \langle$symbol recorded at the other side$\rangle$
10: if receiver:
11: extend T according to incoming symbol

12: **if** either T^R or T^F ends with a '$\leftarrow$' **then**
13: remove the last 4 symbols from T
(and the corresponding transmissions in T^F as well, i.e., its last 2 symbols)
14: output T (a prefix of length n_0)

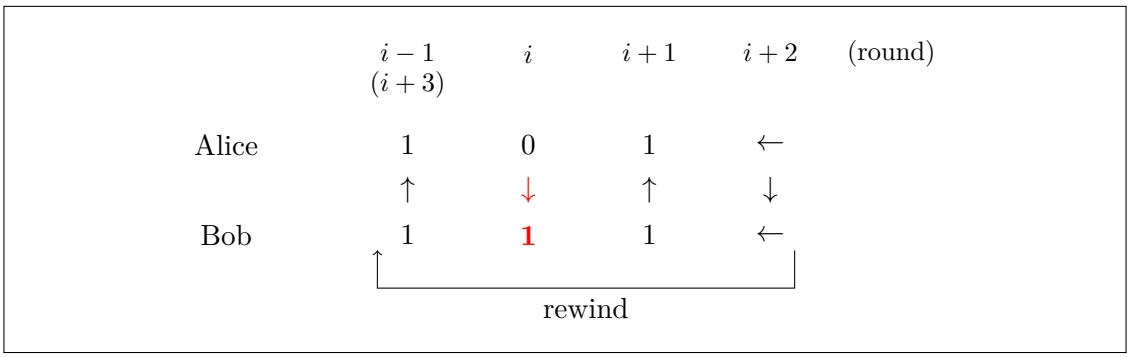

Figure 6.1: Rewinding in Algorithm 7 in the case of a single error at round i. After 3 rounds that contain no errors, the simulation restores to the exact same configuration it had prior to the error.

by preprocessing π_0 so that each party sends a 1 between any two transmissions of the original protocol). In this case, each $\leftarrow$ symbol should rewind the simulation by 6 rounds instead of 4, since it takes 2 bits just to communicate the $\leftarrow$ symbols itself (see Figure 6.2). This binary protocol can tolerate up to a fraction $1/6 - \varepsilon$ of noise.

Theorem 6.2 ([28]). For any $\varepsilon > 0$, there exists an alternating *binary* interactive coding scheme that simulates any protocol π_0 in $O_\varepsilon(n_0)$ rounds over channels with noiseless feedback, and is resilient to $1/6 - \varepsilon$ corruptions.

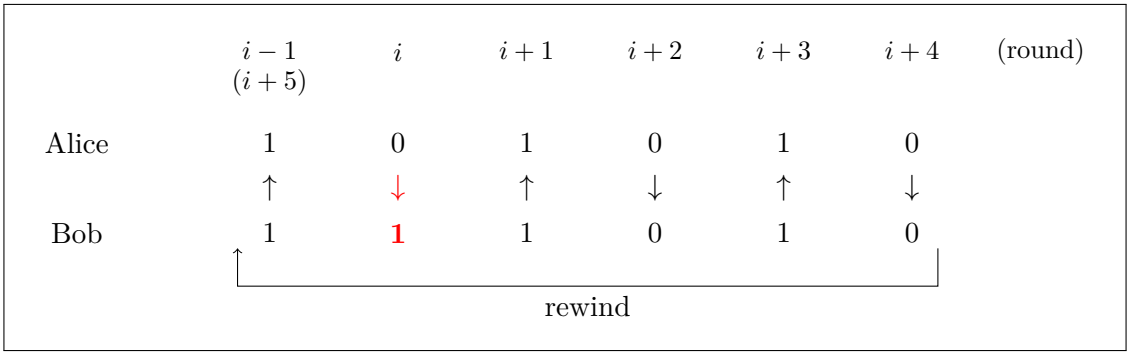

Figure 6.2: Rewinding the binary variant of Algorithm 7, where 00 denotes $\leftarrow$. A single error at round i followed by 5 rounds free of errors restores the simulation to the exact same configuration it had prior to the error.

The resilience of 1/6 and 1/4 achieved by the above schemes is, in fact, tight for coding schemes whose order of speaking is fixed.

Theorem 6.3 ([28]). No interactive protocol for the identity function $f(x, y) = (x, y)$ with a fixed order of speaking can tolerate noise rate of 1/4 with probability greater than 1/2, and no binary protocol with fixed order of speaking can tolerate noise rate of 1/6 with probability greater than 1/2, even in the presence of noiseless feedback.

Since the model of channels with noiseless feedback is stronger than the standard model, the upper bound carries over to the standard case and extends the upper bound of Theorem 2.12 for the standard setting. That is, the upper bound of 1/6 applies also for binary noisy channels without feedback [28].

Proof. The 1/4 bound follows from the same reason as in Theorem 2.12.

The 1/6 bound follows by extending a technique by Berlekamp [7] for showing that a noise fraction 1/3 is maximal in the case of unidirectional message coding over channels with feedback.

Since the protocol has a fixed order of speaking, there exists one party that speaks less than half of the rounds; assume without loss of generality it is Alice and that she speaks for a total of $n_A \leq n/2$ rounds. Assume Alice holds one of three possible inputs a, b, c. Consider the following attack, defined recursively on the rounds i where Alice speaks. At round $i = 1$, the channel considers the bit Alice sends on each of her inputs (denote these bits by $b_a(1), b_b(1), b_c(1)$, respectively). Note that at least two of these bits are the same. The attack changes the bit Alice sends at round $i = 1$ to $\mathsf{majority}(b_a(1), b_b(1), b_c(1))$. That is, for at least two inputs, the channel does nothing, and for the third input, the channel may need to flip the bit. At every other round $i < R$ where Alice

is the speaker (we set R shortly), the channel considers the bit Alice sends on each of her inputs $(b_a(i), b_b(i), b_c(i)$ respectively) given the preceding attack on rounds $[1, i-1]$ and changes Alice's transmission so it becomes $\mathsf{majority}(b_a(i), b_b(i), b_c(i))$.

Denote by $N_a(i)$ the noise the preceding attack causes through round i given that Alice holds the input a (similarly define $N_b(i), N_c(i)$). Set R to be the minimal round so that second-largest value out of $(N_a(R), N_b(R), N_c(R))$ equals $n_A/3$. Without loss of generality, assume c maximizes this value at round R and that b is the second-largest value, that is, $N_b(R) = n_A/3, N_a(R) < n_A/3$. Also note that since the attack makes a single corruption in at most one input every round, then $R \geq N_a(R) + N_b(R) + N_c(R)$. Finally, note that up to round R, Bob sees exactly the same view whether Alice holds a, b, or c. From this point on, we don't care about the input c, as we will show the attack succeeds on inputs a and b.

From round $R+1$ until the end of the protocol (round n_A), if Alice holds a, the channel changes all Alice's transmissions to be what Alice would have sent given that she had the input b. If Alice holds b, the channel does nothing. In Bob's view, exactly the same bits are received between round R and the end of the protocol (and therefore, throughout the entire protocol) whether Alice holds a or b.

We are left to show that the total noise rate on Alice's side is at most $1/3$ (then the total noise fraction is $1/6$, since Alice speaks for at most $n_A \leq n/2$ rounds). If Alice holds b, then the attack stops at round R, when $N_b(R) = n_A/3$, as needed. When Alice holds a, the channel corrupts $N_a(R)$ bits until round R and at most $(n_A - R)$ bits afterwords. Recall that $R \geq N_a(R) + N_b(R) + N_c(R)$ and that $N_b(R), N_c(R) \geq n_A/3$; thus the attack makes at most

$$\begin{aligned} N_a(R) + (n_A - R) &\leq N_a(R) + n_A - N_a(R) - 2 \cdot \frac{n_A}{3} \\ &\leq \frac{n_A}{3} \end{aligned}$$

corruptions, as needed. □

Nonadaptive coding schemes with nonfixed order of speaking

One exceptional property of the model of channels with noiseless feedback is the fact that nonadaptive protocols (Definition 2.3) need not have a fixed order of speaking. Since both parties have a joint view of the *received* symbols, they can determine the next party to speak as a function of the (noisy) received symbols and this determination will be agreed on both sides. This property does not happen in other noise models.

Theorem 6.4 ([19, 28])**.** Any nonadaptive protocol, either over a standard noisy channel or over an erasure channel, must have a fixed order of speaking.

In contrast, protocols over channels with noiseless feedback can have an order of speaking which depends on the noise, while the protocol is still nonadaptive, that is, while there is a consensus regarding the next party to speak (and without the subtleties of adaptive protocols discussed in §4). Yet, similar to the case of adaptive protocols, allowing the parties to change the order of speaking leads to a better noise resilience. Surprisingly, the alphabet size of the channel does not come into play in this setting—a coding scheme with resilience $1/3$ can be shown for channels with binary alphabet as well as larger alphabet [28], and furthermore, this bound is tight!

Theorem 6.5 ([28]). For any $\varepsilon > 0$, there exists a (binary) interactive coding scheme that simulates any protocol π_0 in $O_\varepsilon(n_0)$ rounds over channels with noiseless feedback and is resilient to a fraction $1/3 - \varepsilon$ of corruptions.

It is easy to see how Algorithm 7 can be improved when the order of speaking is not fixed: when $\leftarrow$ is sent, the protocol should rewind only *two* rounds (exactly to the round where the error happened) rather than three rounds. The only purpose of the extra round in Algorithm 7 was to keep the order of speaking alternating! This immediately implies that in the new nonfixed variant, every error stalls the simulation by 3 rounds rather than 4, and thus a noise fraction of $1/3 - \varepsilon$ is achievable. Naively converting this scheme to the binary case immediately gives a noise resilience of $1/5 - \varepsilon$.[1] This, however, can be improved.

The binary coding scheme tries to simulate the preceding behavior, yet it is slightly more complex. The scheme is described in Algorithm 8. The simulation is based on sending messages of varying length, where the adversarial channel needs to corrupt a large fraction of a message to corrupt it, while a smaller fraction of noise being detected leads to ignoring that message[2]. Specifically, each message consists of two parts: a payload that carries the information $\{0, 1, \leftarrow\}$ (corresponding to their meaning in the variant of Algorithm 7 that resists noise $1/3 - \varepsilon$), and *confirmation bits* that inform the receiver whether or not the payload was received correctly. When the payload is received correctly (as learned via the feedback), the sender sends many 1 bits to "confirm" this event. Otherwise, the sender transmits many 0 bits to indicate that the payload should be ignored. These confirmation bits can be tempered with; however, this causes the adversarial channel to invest more and more of its corruption budget in order to corrupt a single payload. In order to confirm a message the number of 1-confirmation bits needs to be substantially larger than the number of 0-confirmation bits. At the same time, if the number of 0-confirmation bits exceed $1/3$ of the current message length, the payload will be ignored. We now formally prove that Algorithm 8 is resilient to the optimal noise level stated in Theorem 6.5 (proof essentially taken from [28]).

Proof. Consider an instance of the protocol that failed to compute the correct output; we will show that the noise in that instance must have been $\geq 1/3 - O(\varepsilon)$.

Let N be the amount of times the protocol executed the loop at line 1, and for $i = 1, \ldots, N$, let m_i be the entire transmission communicated during the i-th instance of the loop (i.e., $|m_i| = \mathit{sentLength}$ when reaching line 19).

Each message m_i can be confirmed or unconfirmed as explained above. If some message m_i is confirmed it can either be *correct* or *incorrect* according to whether or not any of its payload bits msg (i.e., its first two bits), were flipped by the channel. We split $m_1, \ldots, m_N$ into three disjoint sets:

$$\begin{aligned} U &\triangleq \{i \leq N \mid m_i \text{ is unconfirmed}\} \\ C &\triangleq \{i \leq N \mid m_i \text{ is confirmed and correct}\} \\ W &\triangleq \{i \leq N \mid m_i \text{ is confirmed and incorrect}\}. \end{aligned}$$

It is easy to see that an unconfirmed message has no effect on the simulated transcript, because any such message is just ignored by the parties. If a message is confirmed, it can either be inter-

[1] These two schemes also implicitly appeared in [57].

[2] A somewhat similar approach was taken in [1], in order to obtain noise resilience $1 - \varepsilon$ over erasure channels using adaptive coding schemes.

Algorithm 8 A binary coding scheme over channels with feedback [28]

Input: an alternating binary protocol π_0 of length n_0 and an input x. A noise parameter $1/3 - \varepsilon$.

Initialize $T \leftarrow \emptyset$.
$T = (T^S, T^R, T^F)$ is the simulated transcript, split into sent, received and feedback bits

1: **repeat**
2: **if** conditioned on T^F and T^R it is your turn to speak in π_0 **then**
3: $sentLength \leftarrow 2$
4: $conf_0 \leftarrow 0$, $conf_1 \leftarrow 0$
5: **if** $T^S = T^F$ **then** ▷ No corruptions are known
6: $rewind = 0$
7: **else** ▷ The transcript at the other side is corrupt
8: $rewind = 1$
9: $info \leftarrow \pi_0(x \mid T^F, T^R)$ ▷ the next bit of π_0 given the current transcript
10: $msg \leftarrow (info, rewind)$
11: send msg
12: **while** $(conf_0 < sentLength/3)$ and $(conf_1 - conf_0 < 1/\varepsilon)$ **do**
13: **if** msg received correctly **then** ▷ verify via the feedback
14: send 1
15: **else**
16: send 0
17: $sentLength \leftarrow sentLength + 1$
18: let b be the bit received at the other side (learned via the feedback); set $conf_b \leftarrow conf_b + 1$
19: **if** $conf_0 \geq sentLength/3$ **then** ▷ message is not confirmed, ignore
20: **continue** (next loop instance)
21: **else if** $conf_1 - conf_0 \geq 1/\varepsilon$ **then** ▷ msg is confirmed: rewind or advance T according to info/rewind received at other side
22: **if** $\langle rewind$ bit recorded at the other side$\rangle = 0$ **then**
23: $T^S \leftarrow T^S \circ info$
24: $T^F \leftarrow T^F \circ \langle info$ bit recorded at the other side$\rangle$
25: **else if** $\langle rewind$ bit recorded at the other side$\rangle = 1$ **then**
26: remove the last symbol of T^R, T^S, T^F
27: **else** ▷ The other party sends a message
28: record msg, and confirmation bits according to the conditions of the while loop on line 12.
29: if msg is not confirmed (line 19), ignore msg and **continue**
30: if msg is confirmed (line 21):
either extend T^R (if $rewind = 0$) or delete the suffix bit of T^R, T^S, T^F (if $rewind = 1$)
31: **until** n_0/ε^2 bits were communicated

32: output T (a prefix of length n_0)

preted as an information bit or as a rewind request, and the simulation is similar to the ternary algorithm stated above in which after a single incorrect transmission it takes another two correct transmissions in order to revert the simulation to its state just before the corruption has occurred. This immediately implies that the simulation in our case succeeds as long as

$$|C| - 2|W| \geq n_0. \tag{6.1}$$

Next, we bound the length and induced noise rate of a single message.

Lemma 6.6. For any i, $|m_i| \leq 2 + 3/\varepsilon$.

Proof. Assume a message reaches length $2+3/\varepsilon$, and consider its $3/\varepsilon$ confirmation bits: if $1+1/\varepsilon$ of these bits are zeros, then the message is unconfirmed since $(1+1/\varepsilon)/(2+3/\varepsilon) > 1/3$. Otherwise, there are at most $1/\varepsilon$ zeros and at least $3/\varepsilon - 1/\varepsilon \geq 2/\varepsilon$ ones, thus the difference between confirmation zeros and ones is at least $1/\varepsilon$ and the message is confirmed. □

Note that for a confirmed message, $2 + 1/\varepsilon \leq |m_i| \leq 2 + 3/\varepsilon$, and for an unconfirmed message $3 \leq |m_i| \leq 2 + 3/\varepsilon$. Since the total amount of bits the protocol communicates is n_0/ε^2, we have

$$|C| + |W| < \frac{n_0}{\varepsilon}. \tag{6.2}$$

The specific length of a message relates to the amount of corruption the channel must have performed during the communication of that message. For the following analysis, recall that any m_i contains 2 bits of payload, $conf_0$ 0-confirmation bits, and $conf_1$ 1-confirmation bits, that is,

$$|m_i| = 2 + conf_0 + conf_1.$$

Consider the following cases:

- If $i \in C$ then the first two bits of m_i were received correctly, and the channel could have only flipped some of the 1-confirmation bits into 0-confirmation bits. The amount of corrupted bits is exactly $conf_0$. Since the message was eventually confirmed, it holds that $conf_1 - conf_0 = 1/\varepsilon$, and thus $conf_0 = (|m_i| - 2 - 1/\varepsilon)/2$.

- For unconfirmed messages, $i \in U$, the message gets unconfirmed as soon[3] as $conf_0 \geq |m_i|/3$. There are two cases: (i) if the information/control bits are correct, then the noise is any 0-confirmation bit, thus $conf_0 \geq |m_i|/3$; (ii) if the information/control bits are corrupt, then the noise is the corruption of the information/control plus any 1-confirmation bit. For this latter case we have that $conf_0 < |m_i|/3 + 2/3$ and then, $(conf_1 + 2) > \frac{2}{3}(|m_i| - 1)$ which implies that the number of corruptions is at least $(conf_1 + 1) > \frac{2|m_i|}{3} - 5/3$. It is easy to verify that $(conf_1 + 1) \geq \frac{|m_i|}{3}$.

- For $i \in W$, the corruption consists of at least one of the information/control bits and any $conf_1$ received. We have $conf_1 - conf_0 \geq 1/\varepsilon$ thus $conf_1 \geq conf_0 + 1/\varepsilon$ or equivalently $conf_1 \geq (|m_i| - 2 + 1/\varepsilon)/2$. Therefore, the number of corruptions is at least $(|m_i| + 1/\varepsilon)/2$.

[3] It also holds that $conf_0 - 1 < |m_i - 1|/3$, for otherwise the message would have been unconfirmed in the previous round. This implies that $conf_0 < |m_i|/3 + 2/3$.

Therefore, the global noise rate in any given simulation is lower bounded by

$$\text{Noise Rate} \geq \frac{\sum_{i\in C}\frac{1}{2}(|m_i|-2-1/\varepsilon)+\sum_{i\in U}\frac{1}{3}|m_i|+\sum_{i\in W}\frac{1}{2}(|m_i|+1/\varepsilon)}{\sum_{i\in C}|m_i|+\sum_{i\in U}|m_i|+\sum_{i\in W}|m_i|}.$$

We can rewrite the noise rate as

$$\begin{aligned}
&\geq \frac{\frac{1}{2}\sum_{i\in C}|m_i|-\frac{1}{2}|C|(2+\frac{1}{\varepsilon})}{n_0/\varepsilon^2}+\frac{\frac{1}{3}\sum_{i\in U}|m_i|}{n_0/\varepsilon^2}\\
&\quad+\frac{\frac{1}{2}\sum_{i\in W}|m_i|+\frac{1}{2}W\cdot\frac{1}{\varepsilon}}{n_0/\varepsilon^2}\\
&\geq \frac{1}{3}+\frac{\frac{1}{6}\sum_{i\in C}|m_i|-\frac{1}{2}|C|(2+\frac{1}{\varepsilon})+\frac{1}{6}\sum_{i\in W}|m_i|+\frac{1}{2}W\cdot\frac{1}{\varepsilon}}{n_0/\varepsilon^2}\\
&\geq \frac{1}{3}+\frac{\frac{1}{6}\sum_{i\in C}|m_i|-\frac{1}{2\varepsilon}|C|+\frac{1}{6}\sum_{i\in W}|m_i|+\frac{1}{2\varepsilon}|W|}{n_0/\varepsilon^2}-O(\varepsilon),
\end{aligned}$$

where we used Eq. (6.2) for the last inequality. We now use the fact that the simulation instance we consider failed to simulate π correctly. Using Eq. (6.1) we have that $|C|-2|W|<n_0$, or equivalently, $|W|>\frac{1}{2}(|C|-n_0)$. If $|C|<n_0$ it is trivial that the error rate is $\geq 1/3-O(\varepsilon)$. Otherwise, the error rate increases as we increase the length of messages in C and W. Since those messages are confirmed, they are of length $\geq 1/\varepsilon$. Then,

$$\begin{aligned}
&\geq \frac{1}{3}+\frac{\frac{1}{6\varepsilon}|C|-\frac{1}{2\varepsilon}|C|+\frac{1}{12\varepsilon}(|C|-n_0)+\frac{1}{4\varepsilon}(|C|-n_0)}{n_0/\varepsilon^2}-O(\varepsilon)\\
&\geq \frac{1}{3}-O(\varepsilon).
\end{aligned}$$

□

We conclude by showing that 1/3 is an upper bound on the noise any interactive protocols over channels with feedback can tolerate.

Theorem 6.7 ([28])**.** No nonadaptive interactive protocol over channels with noiseless feedback for the identity function $f(x,y)=(x,y)$ can tolerate noise rate of 1/3 with probability greater than 1/2.

Proof. Assume a protocol π of length n for the identity function. We will show an attack that corrupts at most $n/3$ symbols, so that Alice's received symbols (and feedback) look the same for two different inputs of Bob. Then, Alice cannot output the correct value with probability more than 1/2.

Assume Alice holds some input x, and assume Bob holds either y or y'. Consider an uninterrupted instance of $\pi(x,y)$, and assume (without loss of generality) that Bob is the party that speaks for at most $n/3$ rounds between rounds $[1,2n/3]$ in that instance.

Now consider the following attack. When Bob holds y', the channel changes each symbol Bob sends during rounds $[1,2n/3]$ to be exactly what Bob would have sent if he had y. After round $2n/3$,

the attack stops. The crucial observation is that the received symbols in this instance are identical to the ones in the uninterrupted instance of $\pi(x, y)$. From Alice's point of view, the order of speaking must be the same as in $\pi(x, y)$, and since the protocol is nonadaptive, there is consensus regarding this order of speaking. Therefore, Bob still talks at most $n/3$ rounds up to the round where the attack stops. Thus, the attack has a maximal noise rate of $1/3$.

When Bob holds y, the protocol runs until round $2n/3$, uninterrupted. Then, the channel corrupts every symbol Bob sends into the symbol Bob would have sent at the same round under the attack described before. Since we attack only $n/3$ rounds, the maximal error rate is at most $1/3$.

Finally, note that in both attacks the view of Alice is exactly the same (including her feedback, since the channel never corrupts her outgoing transmissions). Thus, Alice cannot output the correct value with probability more than $1/2$ on at least one of (x, y) and (x, y'). □

The above proof assumes a deterministic protocol but can easily be extended to the case of randomized protocols by attacking the party that speaks more times during rounds $[1, 2n/3]$ in expectation.

6.1.2 Rate

Remark 6.1. In this section we assume π_0 is an alternating (binary) protocol. This comes with a certain loss of generality, as discussed in §5.3.

Recall the rewind-if-error paradigm depicted in Figure 2.3. The presence of a noiseless feedback channel greatly simplifies this approach: Step 2 (check if the other side agrees) is immediate, since the feedback channel tells the party whether or not the other side received the correct transmission. Therefore, the immediate candidate for a communication-efficient coding scheme would be the following: (1) communicate k symbols of π_0; (2) check via the feedback if the other's side transcript is correct; (3) send 1 bit to indicate whether the transcript seems consistent; (4) rewind if either side indicated inconsistency, or continue to the next k bits otherwise.

Indeed, the above scheme was proposed by Pankratov [69]. The value of k should be chosen to $O(1/\sqrt{\varepsilon})$ to minimize the communication (see the discussion in §5). This gives to the following,

Theorem 6.8 ([69])**.** For any small-enough $\varepsilon > 0$, there exists a (binary) interactive coding scheme that simulates any alternating protocol π_0 over channels with noiseless feedback with at most a fraction ε of bit flips. The scheme has rate $1 - O(\sqrt{\varepsilon})$.

Gelles and Haeupler [38] show that the rate can be further improved—and provide a coding scheme with rate $1 - O(H(\varepsilon))$—over channels with noiseless feedback, assuming a fraction ε of noise.[4] See Figure 6.3 for a comparison between the optimized rates of the schemes in [69] and [38].

It is interesting to compare the interactive case with the unidirectional case. It is well known that the capacity of a BSC_ε channel is $1 - H(\varepsilon)$ [80] and that the capacity does not change in the presence of noiseless feedback [79]. However as mentioned in §5, the capacity (i.e., the maximal achievable rate) of interactive schemes over a BSC_ε is probably $1 - O(\sqrt{\varepsilon})$, and the scheme of [38] suggests that the presence of feedback *does* change the capacity in the interactive setting.

[4] This implies the same rate over $\mathsf{BSC}_{(1-c)\varepsilon}$ with noiseless feedback, since such a channel makes more than εn bit flips with probability $2^{-\Omega_c(\varepsilon n)}$.

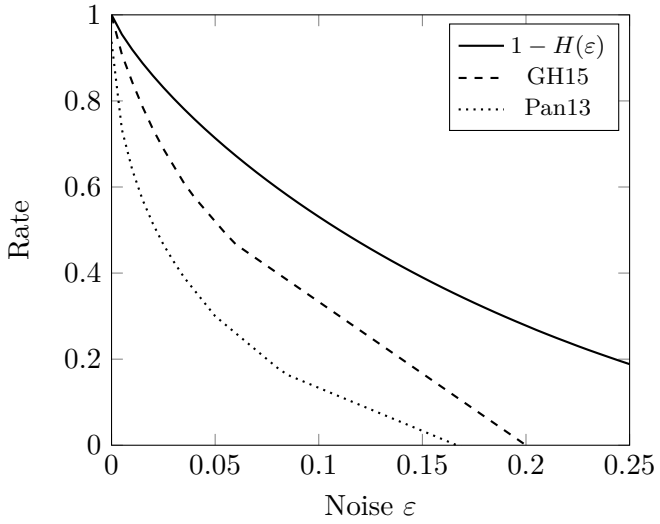

Figure 6.3: A comparison of the optimal rate of the Pan13 scheme [69] and the GH15 scheme [38].

Furthermore, over channels with noiseless feedback, the capacity of the interactive setting and the noninteractive setting are asymptotically the same, $1 - \Theta(H(\varepsilon))$, maybe up to the constant hidden by the Θ notation. Indeed, it is impossible that an interactive scheme (say for the function $f(x, y) = (x, x)$) outperforms the rate of a unidirectional coding scheme that communicates x from Alice to Bob over channels with noiseless feedback. This makes the rate of the scheme of [38] tight (again, up to the constant in the leading term).

The scheme of [38] is given in Algorithm 9. The main ideas that lead to the improved rate are the following. In the scheme of [69], we lose 2 bits (for the "control" data, i.e., the rewind/continue command) per block of k bits of simulation. However, the vast majority of the simulated blocks contain no errors; only a fraction ε of these blocks contain errors. Then, the idea is to send the rewind/continue command only when needed and not every block. To this end, the command is encoded as sending k zeros in a row. That is, whenever a sequence of k zeros is received, the parties remove $2k + 1$ bits from their recorded transcript T and then continue with the simulation. This way, every error delays the simulation by exactly $2k + 1$ rounds,[5] yielding a total communication of $n_0 + \varepsilon n_0(2k + 2)$. Taking $k = O(\log 1/\varepsilon)$ gives the claimed rate.

We need to make sure that the protocol π_0 never sends k consecutive zeros by itself. To this end, the protocol π_0 is preprocessed in the following manner. First, a random string of length $O(\varepsilon n_0)$ is exchanged between the parties. This randomness is then expanded to an almost uniform string of length n_0, for example, via small-biased probability spaces [67, 3], and XORed, bit by bit, to the transcript of π_0. In this preprocessed transcript, any sequence of k bits equals the zero string with probability almost 2^{-k}. Every segment of k consecutive zeros is then broken by adding a 1 after the $(k - 1)$-th zero. This increases the preprocessed transcript (and thus the simulation) by $n_0 2^{-k}$ bits, so that choosing $k = 2c \log 1/\varepsilon$ yields an increase in the communication of $\varepsilon^c n$, which does not affect the obtained rate as long as $c > 1$.

Theorem 6.9 ([38])**.** For any small enough $\varepsilon > 0$, Algorithm 9 efficiently simulates any alternating

[5] We assume that the order of speaking in the simulation needs not be fixed; otherwise one additional round may be wasted.

Algorithm 9 A communication-efficient coding scheme over channels with feedback [38]

Input: an alternating protocol π_0 of length n_0, and an input x. An error parameter $\varepsilon > 0$.
Set $k = \Theta(\log \frac{1}{\varepsilon})$.

procedure $\tilde{\pi}_0$(x) ▷ Preprocessing π_0 to eliminate blocks of k zeros

1: $R \leftarrow$uniform string in $\{0,1\}^{\varepsilon n_0}$ that does not contain k consecutive zeroes
2: send R; receive R'
3: $mask \leftarrow \mathsf{expand}(R, R')$ ▷ $mask \in \{0,1\}^{n_0}$ is $2^{-\varepsilon n_0}$-away from k-wise independent [67, 3]
4: **for** $i = 1$ to n_0 **do** ▷ (if speaker; receiver is symmetric)
5: **if** the last transmitted k bits are all zeros **then**
6: send 1
7: send $\pi_0[i] \oplus mask[i]$

end procedure

▷ Simulating $\tilde{\pi}_0$ of length $\tilde{n}_0$

1: initialize $T \leftarrow \emptyset$; $T^F \leftarrow \emptyset$
T is the simulated transcript viewed so far, split into sent bits T^S and received bits T^R. Let T^F be the characters received by the other side (as learned via the feedback channel)
2: **for** $i = 1$ to $n = \tilde{n}_0(1 + \Theta(\varepsilon \log \frac{1}{\varepsilon}))$ **do**
3: **if** $T^F = T^S$ **then**
4: $T \leftarrow T \circ \tilde{\pi}_0(x \mid T)$ ▷ run one step of $\tilde{\pi}_0$, given the transcript so far is T
5: **else**
6: Send '0' whenever it is your turn to speak; extend T, T^F accordingly
7: **if** T^R or T^F end with k zeros **then** ▷ remove the last $2k+1$ exchanged bits
8: Remove the last $2k+1$ bits from T, and the corresponding transmissions in T^F
9: output T (a prefix of length $\tilde{n}_0$)

binary protocol π_0 over channels with noiseless feedback, has a rate of $1 - \Theta(H(\varepsilon))$, and is resilient to a fraction ε of bit flips with probability $1 - 2^{-\Omega_\varepsilon(n_0)}$.

6.2 Erasure channels

An erasure channel can substitute a transmitted symbol with a special *erasure mark*, denoted by $\perp$. As mentioned, this noise is weaker than the standard noise model since the receiver always knows when a symbol was erased. As with the standard model, one can consider the probabilistic case, where each symbol is erased with some fixed probability ε, or the adversarial case, which limits only the total number of erasures. In this section we first discuss coding schemes that achieve optimal resilience (with a constant rate), and then discuss the case of computationally-efficient schemes.

6.2.1 Noise resilience

The first coding scheme for an adversarial erasure channel was given by Franklin, Gelles, Ostrovsky, and Schulman [35]. That scheme is based on the simulation of Braverman and Rao (Algorithm 1) combined with the well-known fact that erasures are twice as easy to correct than standard errors. Indeed, codes with Hamming distance m can correct up to $m - 1$ erasures, yet only $\lfloor \frac{m-1}{2} \rfloor$ substitution errors. This leads to a scheme that tolerates twice as many corruptions than Algorithm 1 over an erasure channel.

Theorem 6.10 ([35])**.** For any $\varepsilon > 0$, there exists an alternating interactive coding scheme that simulates any protocol π_0 in $O_\varepsilon(n_0)$ rounds over an erasure channel, and is resilient to an erasure fraction of $1/2 - \varepsilon$.

However, the coding scheme of Theorem 6.10 is not efficient, because it is based on tree codes. Efremenko, Gelles, and Haeupler [28] suggested a simple and efficient scheme that also resists a fraction $1/2 - \varepsilon$ of corruptions over erasure channels with alphabet of size 4. The scheme is alternating (Alice sends a symbol in odd rounds and Bob, in even rounds) and is described in Algorithm 10 using an alphabet of size 6 (we later show how to reduce the alphabet size to 4).

The parties progress by simulating the noiseless protocol π_0 bit by bit. The idea of this scheme is that if a party receives some (non-erased) symbol, the party is guaranteed that this is indeed the symbol the other side has sent! Therefore, as long as a bit is received, the receiver can extend its belief in the current simulated transcript T, and this string is always correct. The only issue that may happen is that, due to erasures, the receiver gets a $\perp$ and cannot extend its T. Now, that party needs to be able to tell the other side to retransmit the erased bit. Moreover, the parties need a mechanism that indicates whether a transmitted bit is the next round of π_0 or is a retransmission of a previous round. To this end, the parties can attach to each bit its position in T, but this will increase the alphabet to be polynomially large. Instead, they send a parity mod 3 of the length of their current transcript T. The analysis shows that no matter what error pattern happens, the length of T differs by at most ± 1 between the two parties; thus a parity mod 3 suffices. The scheme is then very simple: a party always sends the next bit according to its current simulated T. If that bit was erased, then the other side cannot extend its T, and the parity of the bit sent back by that party will not match the parity expected by the first party. Thus, the first party cannot extend its T, and it simply retransmits its last sent bit (i.e., the next bit according to its unchanged T).

Algorithm 10 A coding scheme over erasure channels with a small alphabet [28]

Input: an alternating binary protocol π_0 of length n_0 and an input x. A noise parameter $1/2 - \varepsilon$.

```
1: initialize T ← ∅, p ← 0, and m ← (0, 0)
2: for i = 1 to n = ⌈n_0/ε⌉ do
3:     if Sender then
4:         if your turn to speak according to π_0(· | T) then
5:             t_send ← π_0(x | T)          ▷ the next bit of π_0 assuming the transcript T
6:             m ← (t_send, (p + 1) mod 3)
7:             T ← T ∘ t_send
8:             p ← |T| mod 3
9:             send m
10:        else
11:            send the m stored in memory
12:    if Receiver then
13:        record m' = (t_rec, p')
14:        if m' contains no erasures and p' ≡ p + 1 mod 3 then
15:            T ← T ∘ t_rec
16:            p ← |T| mod 3
17: output T (a prefix of length n_0)
```

Theorem 6.11 ([28]). For any $\varepsilon > 0$, Algorithm 10 communicates $O_\varepsilon(n_0)$ bits and is resilient to a fraction $1/2 - \varepsilon$ of erasures.

Proof sketch. A key lemma in the analysis is the fact the difference between the T Alice holds to that of Bob is at most one symbol.

Lemma 6.12 ([28]). let $T_A(i)$, $T_B(i)$ be the variable T held by Alice and Bob, respectively, at round i. For any round $i < n$,

$$||T_A(i)| - |T_B(i)|| \leq 1.$$

Next we show that each error causes at most two rounds where both parties do not extend T. Thus, $n(1/2-\varepsilon)$ erasures lead to at most $n - 2\varepsilon n$ rounds where T does not extend for both parties. In the other $2\varepsilon n$ rounds, T extends for at least one party; thus at the end of the simulation, the sum of the lengths of T at both sides is at least $2\varepsilon n = 2n_0$. Due to Lemma 6.12 the difference in the length at both sides is at most 1, which implies that at both sides, the length of T is at least n_0.

We analyze only the case where the parties were synchronized when the error happened. That is, we assume both hold the same T, say, of length $t = |T|$. Assume an erasure happens at round i, without loss of generality, when Alice is the speaker. Alice's message can be written as (b, p), where b is the t-th bit in the transcript of $\pi_0(x, y)$, and $p = t + 1 \mod 3$. Bob receives $\perp$, and cannot extend his T. Therefore, at round $i + 1$ he sends back a message (b', p') previously sent to Alice (Line 11). It is not difficult to see that b' must be the $(t-1)$-th bit of $\pi_0(x, y)$, and $p' = t \mod 3$. Alice expects a message with parity $t+2 \mod 3$, but instead she receives one with $t \mod 3$; therefore she ignores it. At round $i+2$ Alice retransmits the message (b, p) and the protocol is back on track. The other cases can be analyzed similarly [28]. □

In order to reduce alphabet size, the observation is that, in fact, only four types of messages are needed to implement the above protocol: two messages have the meaning of "I'm one step ahead

of you" or "I'm one step behind you", and the other two messages have the meaning: "we are synchronized and my bit is 0" or "we are synchronized and my bit is 1". This again follows from the fact that the discrepancy in T between the parties is at most ± 1, as stated by Lemma 6.12.

To this end, consider the following preprocessing step for π_0, in which we triple the protocol's length by adding 11 between any two bits π_0 sends. That is, if $\pi_0(x, y)$ is $a_1, b_1, a_2, b_2, \ldots$ (indicating bits sent by Alice and Bob, alternating), then the transcript of the preprocessed protocol would be $a_1, 1, 1, b_1, 1, 1, a_2, 1, 1, \ldots$. It is important to note that all the a_i bits still appear in odd rounds where Alice is the speaker, and all the b_i are in even rounds where Bob is still the speaker. In the preprocessed protocol, both parties know that a "real" bit of information can appear only in transmissions with parity 0 (mod 3). Thus, for the other parities there is no need to send the information bit—it is always 1! Therefore, the following quaternary alphabet suffices for simulating any such preprocessed noiseless protocol:

$$\Sigma = \{0 \times (\text{parity} = 0)\ ,\ 1 \times (\text{parity} = 0)\ ,\ 1 \times (\text{parity} = 1)\ ,\ 1 \times (\text{parity} = 2)\}\,.$$

In order to obtain a binary coding scheme, one can encode each symbol of the quaternary alphabet Σ using a binary code with large relative distance δ. For binary codes of size 4, the maximal distance is $\delta = 2/3$ (e.g., $\{000, 011, 110, 101\}$; see also [8]). The noise resilience obtained by this approach is

$$\left(\frac{1}{2} - \varepsilon'\right) \times \frac{2}{3} = \frac{1}{3} - \varepsilon.$$

Theorem 6.13 ([28]). For any $\varepsilon > 0$, there exists a *binary* alternating coding scheme that simulates any π_0 in $O_\varepsilon(n_0)$ rounds over a binary erasure channel and is resilient to a fraction $1/3-\varepsilon$ of erasures.

It is still unknown whether an erasure rate of $1/3$ is tight for the binary case.

6.2.2 Rate

It is easy to verify that one can obtain a coding scheme for erasure channels with up to a fraction ε of erasures and a rate of $1-O(\sqrt{\varepsilon})$, by building on the scheme of Pankratov [69] described in §6.1.2.

Gelles and Haeupler [38] showed that similar ideas to those used in Algorithm 9 over channels with noiseless feedback can also be used in the case of erasure channels, obtaining a coding scheme with rate $1 - O(H(\varepsilon))$ for a fraction ε of erasures.

Theorem 6.14 ([38]). For any small-enough $\varepsilon > 0$, there exists an efficient randomized coding scheme that simulates any alternating binary protocol π_0 over erasure channels, has a rate of $1 - \Theta(H(\varepsilon))$, and is resilient to a fraction ε of erasures with probability $1 - 2^{-\Omega_\varepsilon(n_0)}$.

The main difference from the coding scheme of Algorithm 9 is that now the receiver needs to notify the sender that a bit was erased, and similar to the case in §6.2.1, multiple erasures may cause the parties to be confused, whether the next transmission is new information or retransmitted information. The solution is to add a small parity that indicates the length of the current simulated T, and helps the parties to distinguish these two cases, similar to Algorithm 10.

In contrast to the setting of channels with noiseless feedback, the capacity of erasure channels in the noninteractive (unidirectional) setting is $1 - \varepsilon$. Therefore, the rate $1 - O(H(\varepsilon))$ obtained by the coding scheme of Theorem 6.14 does not match the upper bound dictated by the unidirectional setting. Nevertheless, it is conjectured that such a rate (up to a constant) is indeed maximal.

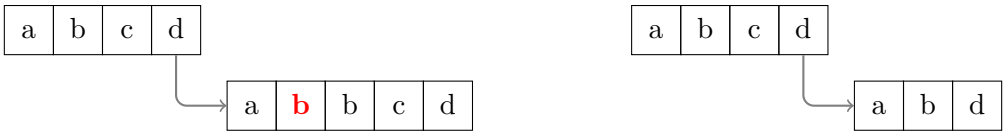

Figure 6.4: An illustration of inserting a b, and deleting a c.

Conjecture 6.1 ([38]). Any nonadaptive coding scheme for alternating binary π_0 over erasure channels with a fraction ε of erasures has a maximal rate of $1 - O(H(\varepsilon))$.

6.3 Channels with insertions and deletions

A channel with insertions and deletion is a channel with a stronger type of noise: in addition to substituting a certain transmitted symbol into a different one, the channel can inject new symbols or completely remove transmitted ones. See Figure 6.4 for an illustration.

We need to be more careful when defining interactive protocols over channels that allow insertions and deletions. One main difference would be the notion of a "round" in the protocol. We cannot assume any external means of synchronization that tells the parties what round they are at, because such a means will allow the parties to detect insertions and deletions as they happen.[6] Therefore, in the insertion/deletion setting a party can determine its "round number" only according to the incoming communication. It follows, that insertions and deletions may cause the parties to get out of synchronization: if the channel inserts a symbol, say, toward Bob, then Bob's round becomes larger by 1 with respect to Alice's round number at the same time. Another problem that may arise in this setting is that a deletion may cause a "denial of service". Assume Alice sends a symbol and then she waits for Bob's reply. If that reply symbol is deleted, then both parties will keep waiting indefinitely for the next symbol to come.

A convenient modeling for interactive protocols in the presence of insertions and deletions was presented by Braverman, Gelles, Mao, and Ostrovsky [17]: they assume that the protocol is alternating and disallow any noise pattern that causes a denial of service. It follows that any corruption must be a deletion followed by an insertion. Such a noise leads to two possible effects: (1) If Alice sends a symbol that gets deleted and Bob gets the inserted symbol, then this is equivalent to a standard substitution of symbols. (2) If Alice sends a symbol that gets deleted and then Alice receives an inserted symbol, then the parties become unsynchronized—Alice's round number is increased while Bob's is not.

Under these assumptions, Braverman et al. [17] show a coding scheme that tolerates up to a fraction $1/18 - \varepsilon$ of insertions/deletions.

Theorem 6.15 ([17]). For any $\varepsilon > 0$, there exists a coding scheme that simulates any protocol π_0 over a channel with insertions and deletions, has a constant rate, and is resilient to a fraction $1/18-\varepsilon$ of corruptions.

The scheme of [17] follows the general tree code simulation approach (i.e., Algorithm 1), yet replacing the tree code with a stronger type of code they call an *edit-distance tree code*. In such

[6] It is easy to see that in the presence of an external means of synchronization, insertions and deletions reduce to erasures.

a tree code, any two suffixes of any two paths are far apart in their *edit distance*, that is, in the number of insertions and deletions it takes to convert the labeling of one of these paths to the labeling of the other path. A key point in the analysis is showing that one can correctly decode the entire sent message as long as the received string is close to the sent one by means of *edit-distance suffix distance*. This notion extends Definition 2.2 to the case of insertions and deletions.

6.4 Quantum channels

Brassard, Nayak, Tapp, Touchette, and Unger [13] extended the discussion of interactive coding into the quantum world. Here two possible models can be considered. The first, introduced by Yao [84], allows the parties to communicate *quantum systems* over a channel with interference. In the second model, introduced by Cleve and Buhrman [23], the parties can only communicate classical bits (over a noisy channel); however they are assumed to preshare an unlimited amount of *entangled* quantum systems. This entanglement allows them to communicate quantum states via teleportation [6]—a quantum phenomena that, given pre-shared entangled states allows to communicate the state of a given quantum system using only classical information. The latter model is as powerful as the former one, since any qubit communicated by the first model can be teleported by the second model and replaced by the communication of two classical bits.

In this model, a noise resilience of up to $1/2 - \varepsilon$ is achievable.

Theorem 6.16 ([13])**.** For any small-enough $\varepsilon > 0$, there exists a quantum coding scheme that simulates any quantum protocol, has a constant rate, and resists a fraction $1/2 - \varepsilon$ of errors.

Surpassing the upper bound 1/4 (Theorem 2.12) is possible due to the fact that the parties preshare a random key that is unknown to the adversary (derived from their shared entanglement); see §2.3.1.

The coding scheme of [13] merges ideas from tree code–based coding schemes [76, 19] and schemes that assume preshared randomness to detect corruptions [35]. The quantum scheme is more complex due to quantum effects that do not allow copying quantum information, thus making it more difficult to "rewind" the protocol. Nevertheless, each party can always reverse the computation by a single step. Through classical communication, the parties can coordinate "reversing" the computation step by step until all errors are corrected.

Open Questions for Section 6

1. What is the maximal noise resilience of *binary* coding schemes over erasure channels?
2. What is the maximal rate of coding schemes over erasure channels? That is, prove or refute Conjecture 6.1.
3. Design *deterministic* coding schemes that achieve a rate of $1 - O(H(\varepsilon))$ over either channels with feedback or erasure channels.
4. What is the maximal noise resilience of coding schemes over channels with insertions and deletions?
5. How can we obtain efficient coding schemes over channels with insertions and deletions?

Note: Some progress has been made on the above open questions. See Addendum in §A.

7

Multiparty Interactive Communication

In this section we augment the setting of interactive communication to include the case of multiple parties performing some distributed computation over an arbitrary network, where the communication links are noisy.

We abstract the communication network as an undirected graph $G = (V, E)$, with $m = |V|$ nodes and $l = |E|$ edges, where each node is a party and each edge (u, v) is a (private) communication link connecting the parties u and v (u and v are said to be *neighbors* in this case). Each party begins with a private input x_i, and the parties' joint goal is to output a function of their joint inputs $f(x_1, \ldots, x_m)$. As before, we assume the existence of a noiseless protocol π_0 that performs the computation in n_0 rounds, assuming the network is noiseless. We seek a coding scheme π that performs the same task as π_0 over a noisy network. As in the two-party case, the noise can be random, that is, each link is an independent BSC_ε, or it can be adversarial, so that the only restriction on the noise is the total number of corruptions that happen during the protocol, throughout the entire network.

In the multiparty case we distinguish two settings of message passing: *synchronous* and *asynchronous*. The synchronous message passing extends the two-party case in a natural way: the protocol progresses by distinct simultaneous steps (rounds), where at each round, every party sends a single message through each of the communication links connected to it. A message here is a single symbol out of the channels' alphabet Σ.

In the asynchronous setting there is no notion of rounds or time. When a message is sent, it is guaranteed to arrive eventually at the other side (but the delay may be arbitrary); it is also guaranteed that over the same link, messages arrive in the same order they were sent. The protocol begins by triggering all parties simultaneously with a vacuous message. Each incoming message may trigger the receiver to send messages to (some of) its neighbors. The protocol ends when all messages are delivered and there is no party that is scheduled to send any further message. A message here may be of any length and is not restricted to a certain alphabet.

In both the synchronous and asynchronous settings, the communication complexity of the protocol is the total number of bits sent throughout the protocol. That is, for a synchronous protocol π (with $|\pi| = n$ rounds), the communication is $n \cdot 2l \log |\Sigma|$. In the asynchronous case, it is the total

lengths of messages sent during the protocol.

7.1 Coding schemes for networks with random noise

The first coding scheme over networks with random noise (BSC_ε) in the synchronous message-passing model was given by Rajagopalan and Schulman [73], extending Schulman's original work to the multiparty case.

In this scheme, each two neighboring parties communicate with each other using a tree code. The content that each party communicates is a simple rewind-if-error simulation that depends on the view of that party, given all the communication from its neighboring parties. That is, the parties exchange symbols from $\{0, 1, \leftarrow\}$, where 0 and 1 indicate an information bit, and $\leftarrow$ instructs the recipient party to delete the latest received information bit.

The main idea is the following: at each round, a party decodes all the communication its neighbors have sent him so far (where messages are encoded using tree codes, separately per neighbor). Based on the current decodings, the party verifies that its outgoing communication so far is consistent with its current understanding of the incoming communication. That is, the party verifies that at each round so far, its sent information bit corresponds to the information the same party should have sent in the noiseless protocol π_0, given the same incoming information. Note that since the *received* transcript changes according to the tree code decoding algorithm, previously sent information bits may be inconsistent given the *current* tree code decoding. If all the sent information is consistent with the current received information, the party sends the next information bit to its neighbors, according to the noiseless protocol π_0. If there is some inconsistency, the party sends the special back symbol that tells the receiver to disregard the last (undeleted) bit.

The scheme is described (for a specific party p) in Algorithm 11. It runs for $2n_0$ rounds, and communicates $O_\varepsilon(\log d+1)$ bits per round, where d is the maximal degree of a node in the network. Specifically, each bit in $\{0, 1, \leftarrow\}$ is encoded via a tree code with distance 8/10. The output symbol is communicated via a repetition code with $k = O_\varepsilon(\log(d + 1))$ repetitions. The exact value of k is determined so that a single symbol is decoded correctly via the repetition code with high probability, that is, of a magnitude $1 - \mathrm{poly}(1/(d + 1))$.[1] In contrast to the repetition code (which depends on the capacity of the BSC_ε channel), note that the distance of the tree code is constant and independent of ε. Specifically, the alphabet size of the tree is a small constant, which is independent of ε. The rate of the scheme is then $O_\varepsilon(1/\log(d + 1))$.

Theorem 7.1 ([73])**.** Given any network with maximal degree d, Algorithm 11 simulates any protocol π_0 over BSC_ε with probability $1 - m2^{-\Omega_\varepsilon(n_0)}$, and rate $O_\varepsilon(1/\log(d + 1))$.

Proof. We first need to distinguish between two types of errors: errors that add up and errors that don't add up. Assume that a single error happens at the incoming message of some party p, say, at round i. At the next round $i+1$, the party may realize that the symbol of round i was incorrect (namely, the tree code decodes a different prefix), and p decides to backtrack by sending $\leftarrow$ to all his neighbors. At round $i + 2$ each of p's neighbors receives a $\leftarrow$ from p and deletes the last undeleted bit received from p. Now, however, due to deleting an incoming bit, the symbols that

[1] Recall that repetition code with k repetitions fails when $\geq k/2 + 1$ copies were received incorrectly. If each symbol is corrupted with independent probability ε, then the decoding fails with probability at most $(4\varepsilon(1 - \varepsilon))^{k/2}$; see, e.g., [36].

Algorithm 11 The Rajagopalan-Schulman coding scheme [73]

Input: a binary alternating protocol π_0 and an input x.

Assume a ternary tree code $\mathcal{T}$ with distance 8/10 known to all parties. Let d be the maximal degree in the network. Thus, each party has at most d different neighbors. Each link is assumed to be a BSC_ε. Set $n = 2n_0$.

for each neighbor j, initialize $s_j[1..n], r_j[1..n] \leftarrow \emptyset$.
for round $i = 1$ to n **do**
  for each neighbor j **do**
    $m_j \leftarrow \mathsf{TCdec}_{\mathcal{T}}(r_j[1..i])$
  if for all neighbors j, $\mathsf{parse}(s_j[1..i])$ is consistent with the view $\{\mathsf{parse}(m_j)\}_j$ **then**
    set $s_j[i]$ as the next bit sent to party j in π_0 given the received transcript so far is $\{\mathsf{parse}(m_j)\}_j$
  else
    for all neighbors j, set $s_j[i] =$ '$\leftarrow$'
  for each neighbor j, send j the last symbol of $\mathsf{TCenc}_{\mathcal{T}}(s_j[1..i])$ (via a $O_\varepsilon(\log(d+1))$-repetition code).
  receive from each neighbor j, a symbol $r_j[i]$ (via repetition code)
output $\mathsf{parse}(s_j[1..n]), \{\mathsf{parse}(m_j)\}_j$ (a prefix of length n_0 per neighbor)

The procedure parse on a string $x \in \{0, 1, \leftarrow\}^*$ scans the string from left to right, and whenever it hits a $\leftarrow$, it deletes that symbol as well as the symbol immediately before it (if such exists). Then it continues with the scan until it reaches the end of x. Example: if $x = 00\leftarrow 11101\leftarrow\leftarrow 00$, then $\mathsf{parse}(x) = 011100$.

were communicated at round $i+1$ are possibly incorrect, and that neighbor needs to send $\leftarrow$ to all *his* neighbors until the entire network backtracks one step of the simulation.

Now assume that two errors happen. We first demonstrate a case in which the two errors do not "add up" and cause only a single step of backtracking throughout the network. Assume that the first error happens again in the incoming message of p at round i. The second error also happens at round i; however it affects the incoming message of party q (which is p's neighbor). Say that at round $i+1$, both p and q realize that an error has happened and both decide to backtrack. In this case, when q receives a $\leftarrow$ from p at round $i+2$, it doesn't need to backtrack again since it has already backtracked one step at round $i+1$. Namely, the bits that q sent that depend on the incorrect bit of p were already removed, and the simulation can continue without any further delay.

The other case, in which errors add up happens, for instance, if the second error affects the communication of p at some round $i' > i$, *after* p has already backtracked or before p realizes it needs to backtrack (so that p keeps sending incorrect bits that should be deleted). Similarly, if the second error happens at one of p's neighbors after that neighbor has performed the backtracking due to receiving $\leftarrow$ from p, (i.e., after round $i+1$), that error will cause an additional backtracking. We can define the *influence cone* of a specific error that happens at party p at round i, as all the pairs (p', i') so that p and p' are connected by a path of length ℓ such that $\ell \leq i' - i$. All these p' will backtracking due to the error at (p, i) after at most ℓ rounds. Any error that happens after that time may cause a secondary backtracking. Yet all errors that happen beforehand will be corrected by the same backtracking and will not add up. We say that several errors $\{e_j = (p, i)_j\}_j$ are in the *same influence cone* if, for any $j' > j$, $e_{j'}$ is in the influence cone of e_j.

Note that in the preceding argument an "error" is the event that the tree code failed to decode (rather than an arbitrary error in the channel, even if the repetition code failed); we emphasize this

fact by calling a tree code failure a *tree error* and a repetition code failure a *repetition error*. The analysis of [73] shows that the simulation succeeds as long as the number of tree errors that "add up" is at most $n_0/2$. Basically, this argument shows that each error causes two rounds of delay in simulation: the round of the error and another round for backing up.

Theorem 7.2 ([73]). If a party at round i simulated a correct prefix of length $j < i$ of the transcript with all its neighbors, then there must have been a sequence of at least $(i-j)/2$ tree errors in the same influence cone.

We run the simulation for $2n_0$ rounds, and the simulation fails if the correct transcript's prefix of some party is shorter than n_0. Theorem 7.2 indicates that $n_0/2$ tree errors that add up must have occurred in the same influence cone. Assume that the tree code has distance α; then $n_0/2$ tree code failures imply at least $\frac{\alpha}{8}n_0$ repetition errors (i.e., the reception of an incorrect symbol $r_j[i]$). This follows from Lemma 3.6: each time the tree code fails, consider the incorrect decoded suffix and let $[r_1, r_2]$ be the interval that describes the rounds that correspond to this suffix. Using Lemma 2.4 we know that the number of repetition code errors within this suffix is at least $\frac{\alpha}{2}(r_2 - r_1 + 1)$. Furthermore, the union of all these intervals is at least $n_0/2$, and Lemma 3.6 tells us there is a set of disjoint intervals with total length at least $n_0/4$. This implies that the repetition code must have failed for at least $\alpha n_0/8$ times. We can fix $\alpha = 8/10$ and determine that the simulation fails only if there are $n_0/10$ repetition errors in a same influence cone.

We can bound the probability of having so many repetition errors at the same influence cone by the following. Given any endpoint $(p, 2n_0)$, there are at most $(d+1)^{2n_0}$ different ordered sequences of $2n_0$ pairs (party,round) in the same influence cone, that is, if (p_1, i_1) comes before (p_2, i_2) in the ordered sequence, then p_1 and p_2 are connected by a path of length at most $i_2 - i_1$. For each such sequence, there are at most 2^{2n_0} possible error patterns with at least $\frac{1}{10}n_0$ errors. Taking a union of all the m possible endpoints $(p, 2n_0)$ of the influence cone, we find that the probability of having at least $\frac{1}{10}n_0$ errors at the same cone is bounded by

$$m(d+1)^{2n_0} \cdot 2^{2n_0} \cdot (p_{\mathsf{rep}})^{n_0/10},$$

where p_{rep} is the probability the repetition code fails. It is easy to see (Footnote 1) that by taking $\mathrm{poly}(\varepsilon, \log(d+1))$ repetitions, we can lower the failure of the repetition code to $p_{\mathsf{rep}} < \left(\frac{1}{2(d+1)}\right)^{20}$. In this case, the simulation succeeds with probability $1 - m2^{-\Omega(n_0)}$. □

Coding schemes with a constant rate for specific topologies.

The scheme of Rajagopalan and Schulman (Algorithm 11) implies a coding scheme with a constant rate for many topologies. Specifically, a constant-rate coding is achievable over graphs with a constant maximal degree $d = O(1)$, for example, a line, a cycle, or $O(1)$-regular graphs. On the other hand, if the maximal degree is super-constant, the rate is vanishing. In the worst case (e.g., over a complete graph or a star), the obtained rate is $O(1/\log m)$.

A scheme by Alon, Braverman, Efremenko, Gelles, and Haeupler [2] obtains a constant-rate coding for another set of topologies not covered by Algorithm 11. Specifically, their scheme obtains a constant rate over any d-regular graph that has "many" short disjoint paths between any two parties. This is formulated via the mixing time of the graph,

Definition 7.1 (mixing time)**.** The mixing time of a graph G is the smallest number t such that a random walk of length t ends at every node in G with probability $\frac{1}{m} \pm \frac{1}{2} \cdot \frac{1}{m}$.

Theorem 7.3 ([2])**.** For any $\varepsilon < 1/2$, there exists a synchronous multiparty interactive coding scheme that simulates any binary protocol π_0 over a d-regular graph G in $O_\varepsilon(n_0 \cdot t^3 \log t)$ rounds, where t is the mixing time of G. The scheme uses a binary alphabet and succeeds with probability $1 - 2^{-\Omega(d^{\Omega(1)} n_0)}$, assuming each channel is a BSC_ε and $d \geq \log^{1+\Omega(1)} m$.

Theorem 7.3 implies a constant-rate coding scheme for any d-regular topology with a constant mixing time $t = O(1)$. These include the complete graph and random d-regular graphs with $d = m^\alpha$ for some constant $\alpha > 0$ (note that if t is the mixing time, then $2d^t > m$). Yet, some topologies are not covered by either the scheme of [73] or by [2]—for instance, a star.

The scheme of [2] builds on Algorithm 11. As shown in the analysis of Theorem 7.1, in order to guarantee the success of Algorithm 11 one needs to guarantee that each transmitted symbol is received correctly at the other side with probability at least $1 - \mathrm{poly}(1/(d+1))$. The scheme in [2] shows how to deliver *all* the symbols of a single round in π_0 in a reliable way that fails with a negligible probability. This, along with the reasoning of Theorem 7.1, yields the coding scheme guaranteed by Theorem 7.3.

Consider the following *neighborhood connectivity* task, which is equivalent to a single round in the noiseless protocol π_0.

Definition 7.2 (neighborhood connectivity)**.** Assume that for each neighboring parties $(u, v) \in E$, party u holds a bit $b_{u \to v}$ which should be sent to v, and the party v holds a bit $b_{v \to u}$ which should be send to u. The neighborhood connectivity task amounts to the computation after which each party u outputs bits $\{b_{j \to u}\}$ for all j's.

Clearly, the neighborhood connectivity task can be solved via a *single* round of *noiseless* communication over the network. Interestingly, coding for multiparty interactive protocols reduces to solving the neighborhood connectivity task:

Corollary 7.4 ([73, 2])**.** Given a scheme that solves the neighborhood task on graph G in k rounds with probability $1 - p$, there exists a coding scheme that simulates any π_0 over G in $O(kn_0)$ rounds and succeeds with probability $1 - m(2(d+1)p)^{\Omega(n_0)}$, assuming each channel is a BSC_ε.

The above is a corollary of the analysis of Theorem 7.1.

The scheme in [2] solves the neighborhood task over d-regular graphs with mixing time t in $O(t^3 \log t)$ rounds and success probability $1 - tm^2 2^{-\Omega(d^{\Omega(1)})}$. With the above corollary, this in turn proves Theorems 7.3. Below we give a brief summary of how to solve the connectivity task over d-regular graphs in a constant number of rounds.

Proof sketch of Theorem 7.3. The scheme for the neighborhood connectivity task is based on the following claims/steps:

1. *For any (arbitrary) list of $L = \{(s_i, d_i)\}$ of source nodes s_i and destination nodes d_i, where any node $u \in V$ appears at most $O(d/t)$ times in L, there exists a list P that contains $|L|$ edge-disjoint paths with length at most $2t$, such that each for any i, the i-th path in P connects s_i and d_i.*

The existence of the list follows from the Lovász local lemma [32, 4]: since the mixing time is t, there are $O(d^{2t}/m)$ paths of length $\leq 2t$ connecting each (s_i, d_i). Additionally, if we randomly pick paths, the probability that two such paths share an edge is $O(m^2/d^{4t})$. The local lemma shows it is possible to choose one path for each pair (s_i, d_i), so that all the paths are jointly edge disjoint.

2. *Let Λ be a parameter. Assume that each party in G needs to send a total number of at most d bits to at most d/Λ different parties in G. Then, there exists a coding scheme over the noisy network G that delivers all the bits in $O(t^2 \log t)$ rounds with probability $1 - n^2 2^{-\Omega(\Lambda)}$.*

   Consider some party p, and assume that for any $0 < i \leq d/\Lambda$, p needs to send a_i bits to some party p_i, where $\sum_i a_i \leq d$. The proof follows by encoding all the bits directed to p_i using a standard error correcting code of length $O(a_i + \Lambda)$. Then, all these codewords are sent to their destinations using the disjoint paths of Step 1. Note that the total number of bits each party needs to send (after the encoding) is $\sum_i O(a_i + \Lambda) = O(d)$; therefore, t instantiations of Step 1 suffice to communicate all the codewords to their destinations. Each instance of Step 1 takes $t \log t$ rounds to send 1 bit along each path of length t (using repetition code of length $\log t$ per edge). Therefore, Step 2 takes a total of $O(t^2 \log t)$ rounds to communicate all codewords.

   Recall that in the neighborhood connectivity task over a d-regular graph, each party begins with d bits targeted to all its d neighbors (1 bit per neighbor). If we were to use Step 2 directly, this would imply $\Lambda = 1$ and a success rate of $1 - O(n^2)$... To maintain a high success rate, we need, say, $\Lambda = d^c$ for some constant $c \in (0, 1)$. However in that case Step 2 cannot be used directly, because it limits the number of different parties p can talk with to $d/\Lambda = d^{1-c}$. The last step in the proof shows how to reduce the neighborhood connectivity task to $O(t)$ instances of Step 2, delivering d bits to at most d/Λ parties at each instance.

3. *The neighborhood connectivity can be solved by $O(t)$ applications of Step 2, with probability $1 - mn^2 2^{-\Omega(\Lambda)}$.*

   The proof goes in a recursive manner, where at each step the network is being split into $O(d/\Lambda)$ disjoint subsets, where the communication happens only among parties within the same subset. Splitting each subset is done via the coding of Step 2: each party sends all the bits that belong to a specific subset to a designated member of that subset (the nontrivial part is to show how to choose these designated parties without violating the conditions of Step 2). With the choice of $\Lambda = d^c$ for some $c < 1$, repeating this "localization" step for about $t/(1-c)$ times reduces the size of each individual subset to

$$m \left(\frac{\Lambda}{d}\right)^{t/(1-c)} \leq d^t \left(d^{-(1-c)}\right)^{t/(1-c)} \leq \frac{d}{\Lambda}$$

   (recall that $2d^t > m$); thus a final application of Step 2 in parallel in every subset completes the task.

□

7.2 Coding for networks with adversarial noise: Upper bounds on the maximal noise

Similar to the two-party setting, the case of adversarial noise is more difficult to handle also in the multiparty setting. Here, rather than a noise that has a "smooth" distribution over all the transmissions (as in the stochastic case), the noise can target specific messages, specific links, or specific parties.

In fact, this puts a stringent constraint on the noise level—it must not be the case that the noise is able to completely "silence" any single party. Indeed, Jain, Kalai, and Lewko [56] observed that no protocol can tolerate more than a fraction $\Theta(1/m)$ of adversarial noise. This bound also holds in the case of adaptive protocols [46, 1], where the number of bits a party sends may depend on the noise.

Theorem 7.5 ([56]). No protocol for the m-party identity function over a network G can output the correct value with probability greater than $1/2$ in the presence of a fraction $4/m$ of adversarial noise.

Proof. The claim is quite trivial in the synchronous setting, where the party with the minimal number of neighbors talks less than $\mathsf{CC}(\pi)/m$ and its entire communication can be corrupted.

Yet this holds also in the asynchronous setting. Let p be the party that, assuming no errors, sends and receives at most $2\mathsf{CC}(\pi)/m$ bits (say, on input x, but without loss of generality, we can assume the same holds for another input x'). The adversary can completely corrupt the view of p, replacing it with the view p would have on input x'. This attack cannot be detected by any party and leads to corrupting at most $4\mathsf{CC}(\pi)/m$ bits, i.e., a fraction $4/m$ of adversarial noise. □

7.3 Coding schemes for networks with adversarial noise: Synchronous setting

Hoza and Schulman [55] considered the case of coding schemes over networks $G = (V, E)$ with $m = |V|$ parties and $l = |E|$ communication links in the synchronous message passing setting assuming adversarial noise model. They show, through a careful analysis of Algorithm 11, that the same coding scheme works even in the presence of a fraction $\Theta(1/l)$ of adversarial noise.[2] That is, the noise resilience is inversely proportional to the number of *edges* in the network.

Theorem 7.6 ([73, 55]). For any graph G, there exists a synchronous coding scheme that simulates any noiseless binary protocol π_0 in $O(n_0)$ rounds, uses a binary alphabet, and is resilient to a fraction $\Theta(1/l)$ of adversarial noise.

In order to improve the noise resilience and obtain the maximal noise resilience, Hoza and Schulman perform a routing that reduces the number of edges that are actually used to $O(m)$. By this, they obtain a higher noise resilience, namely, of $O(1/m)$. More specifically, they take the following course of action:

1. Find a subgraph $G' = (V, E')$ of G with at most $|E'| = O(m)$ edges, so that in any cut $U \subset V$ the number of edges crossing the cut in G' is at least $5l/m$ times the number of edges crossing the cut in G. Formally, let $\delta(U)$ be the set of edges in G crossing U, and $\delta'(U)$ be the set of

[2] By saying that a scheme is resilient to a fraction $\Theta(\xi)$ of noise, we mean that there exists a constant $\varepsilon > 0$ such that the scheme is resilient to a fraction $\varepsilon\xi$ of noise.

edges in G' crossing U, then

$$\frac{5l}{m}\delta'(U) \geq \delta(U).$$

The existence of such a subgraph is guaranteed by [26].

2. Given a subgraph G' with the preceding properties, they show that the neighborhood connectivity task of G can be solved in $O(\frac{l \log m}{m})$ noiseless rounds over a subgraph $\tilde{G}$ that contains the edges of G' and at most $O(m)$ other edges in G. This part involves routing and scheduling of messages performed via multicommodity flow methods [61, 63].

3. Preprocess π_0 so that it communicates only using the edges of $\tilde{G}$. Using Step 2, the preprocessed protocol takes $O(\frac{l \log m}{m} \cdot n_0)$ rounds.

4. Use Theorem 7.6 (Algorithm 11) to obtain a coding scheme that simulates the preprocessed protocol over the graph $\tilde{G}$.

However, in the subgraph $\tilde{G}$ the number of edges is only $O(m)$, rather than l as in G; thus Algorithm 11 provides the preprocessed protocol with resilience of $\Theta(1/m)$. This approach leads to the following.

Theorem 7.7 ([55]). Given any graph $G = (V, E)$ with $|V| = m$ and $|E| = l$ and a noiseless protocol π_0 over G, there exists a synchronous coding scheme that simulates π_0 in $O(\frac{l \log m}{m} n_0)$ rounds over a subgraph $\tilde{G} = (V, \tilde{E})$ and is resilient to a fraction $\Theta(1/m)$ of adversarial noise.

Note that the coding scheme is defined over a different graph $\tilde{G}$ than the original protocol G. This can be seen as a hybrid of the asynchronous model described in the next section: several edges are being "silenced", while over the other edges, a single bit is sent in every round. While a noise resilience of $\Theta(1/m)$ is maximal in the multiparty case (see §7.2), a remaining open question is whether the obtained rate of $O(m/l \log m)$ is optimal in the adversarial-synchronous case and, if so, which topology satisfies it.

In addition to the above, the work [55] considers also the case where the network is given by a *directional* graph. In this case, certain relations between noise tolerance and the topology of the network are given. Other settings that are considered by [55] include the case where the noise is budgeted *per edge* rather than globally over the entire network.

7.4 Coding schemes for networks with adversarial noise: Asynchronous setting

Jain, Kalai, and Lewko [56] gave the first coding scheme assuming asynchronous message passing. Their scheme assumes there exists a party that is connected to all the other parties (i.e., the graph G contains a star), and is resilient to a fraction $\Theta(1/m)$ of adversarial noise.

Theorem 7.8 ([56]). Given any G that contains a star, there exists an asynchronous coding scheme π that simulates any noiseless protocol π_0 over G. The coding scheme communicates $\mathsf{CC}(\pi) = O(\mathsf{CC}(\pi_0))$ bits and is resilient to a fraction $\Theta(1/m)$ of adversarial noise.

The main idea behind the coding scheme [56] is the following. All the communication is sent over the edges of the star subgraph. That is, when p_1 needs to send a message to p_2, it actually

sends a message to the star center (which we simply name p), and then p relays the message to p_2. This way, p has a *global* view of the progress of the protocol.

The communication over each edge (p, p_i) is encoded via a ternary tree code sending the messages $\{0, 1, \leftarrow\}$ and using a simple rewind-if-error paradigm, similar to the case in [73]. Since all the communication goes through the center p, in case one of the parties backtracks, p can backtrack the entire protocol (i.e., all affected parties) until all the parties have removed the incorrect suffix.

Let us give an example that clarifies this idea and highlights several subtleties we need to address. Recall that in the asynchronous message-passing model, a party sends a message only after the event of receiving at least one message. Consider the following noiseless π_0: for 100 times, p_1 sends a bit to p_2 and then receives a bit from p_2. Then, p_1 sends a bit to p_3 and receives a bit back; the same pattern (of $100 + 2$ exchanges) is then repeated again and again.

Assume that the bit sent at the 101st exchange from p_1 to p_3 is corrupted by the channel. As a result, the bit p_3 sends back to p_1 is incorrect and the 100 bits exchanged between p_1 and p_2 afterward are also incorrect. This error cannot be corrected until enough messages are transmitted from p_1 to p_3, allowing the tree code between them to recover the error. This means that only after approximately 300 exchanges occurred in the protoocol, will p_3 realize that the bit received at the 101st exchange is incorrect, and only then would p_3 indicate that its reply to p_1 should be corrected. Only then can the backtracking process in the network begin. However, by that time a suffix of approximately 200 bits between p_1 and p_2 needs to be deleted.

This example suggests it may take a while to correct errors that happen between parties that "do not speak a lot", and until such errors are corrected, the protocol keeps communicating large numbers of (incorrect) bits, which are wasted.

To mitigate this effect, the scheme of [56] *polls* each party from time to time, even if this party is not scheduled to speak in the underlying π_0. That is, assume the alphabet $\{0, 1, \leftarrow, H\}$, where the symbol H (*hold*) causes the noncenter recipient only to reply with an H and is ignored otherwise. The center allows the simulation to progress in chunks of $O(m)$ bits. After each such chunk, the center sends an H to each party (and expects to receive an H back). This way, every party has a chance to "talk" at a constant frequency, and errors are being detected in a timely manner, that is, within $O(m)$ bits of communication.

The analysis in [56] shows that running the preceding coding scheme for $6n_0$ rounds (assuming a tree code with distance $1/2$) suffices to complete the protocol correctly as long as the fraction of bit errors does not exceed $\frac{1}{10638m}$. More specifically, the analysis plots a relation between the progress of the simulation and the number of errors, and shows that in order to delay the progress by xn "rounds", it must be that the tree code failed in decoding $O(x)$ symbols (where the constant depends on the distance of the tree and the amount of rounds it takes to backtrack after a single error has happened). Tree code failures amount to corrupted symbols, as explained in the proof of Theorem 7.2.

Balancing the communication. The work of Lewko and Vitercik [62] follows the same idea of [56], but instead of having a designated party that serves as the coordinator, this role is being distributed among all the parties in a round-robin fashion. Specifically, the protocol is run in chunks, where during the i-th chunk, the i-th party p_i serves as the coordinator. Hence, the topology is assumed to be a complete graph.

Between each two chunks, the coordinator performs a consistency check, where every party sends the coordinator a hash of the view that party holds. The coordinator verifies that the hash values match its own view and either instructs the parties to continue to the next chunk or to rewind one chunk. In addition, p_i passes its role as a coordinator to the next party, p_{i+1}. Note that the new coordinator has only its own local view on the computation so far and cannot perform the consistency check without the "global view" that p_i has. To this end, the old coordinator sends the new one hash values of its entire view (separately per party). These hashes will serve the new coordinator as the "global view", and will be concatenated to the communication in the new chunk. [Each party does the same: if the party passes the consistency check, then its view matches the hash sent to the new coordinator. In the next chunk the party can hash its communication of the last chunk concatenated with the hash of the communication up to that chunk.]

This scheme achieves a similar rate and resilience as [56]; however, the net effect of distributing the role of the coordinate is avoiding a "bottleneck" that concentrates all the communication. That is, while in [56] the center's view consists of all the $\mathsf{CC}(\pi)$ bits communicated during the protocol, in [62], the view of each party consists of $O(\mathsf{CC}(\pi)/m)$ bits.

Open Questions for Section 7

1. What set of topologies admits a synchronous coding scheme with rate $O(1)$ over a BSC_ε? In particular, find a precise relation between a graph G to the maximal rate it admits.
2. What is the maximal rate obtainable for synchronous coding schemes that are resilient to a fraction $\Theta(1/m)$ of noise? (That is, is the rate $O(m/l \log m)$ of Theorem 7.7 optimal?)
3. Devise a coding scheme in the asynchronous model for a general topology (specifically, one that does not have a star as a subgrph). What is the best rate that can be achieved with maximal noise?
4. Find a coding scheme that is resilient to a fraction $\Theta(1/m)$ of noise, where the topology is *not* known in advance. What is the maximal rate in this case?
5. Extend the above coding results to other communication and noise models: specifically, the case of noisy Broadcast or Multicast channels, or the case of point-to-point channels with insertion and deletion noise.

Note: Some progress has been made on the above open questions. See Addendum in §A.

8

Applications and Related Topics

In this section we survey several applications of noise-resilient interactive protocols and several other topics that relate to noise-resilience techniques presented in this manuscript.

8.1 Noise-resilient formulas

Kalai, Lewko, and Rao [57] showed how to use coding for interactive communication in order to fortify formulas (i.e., circuits with fan-out 1) against a specific type of faulty gates, namely, gates that may short-circuit one of the inputs to the output. More generally, they show how to compile any formula F into a resilient formula E with size $|E| = \text{poly}(|F|)$, so that E is correct even if at any path between input and output at E, a fraction of up to $1/10 - \varepsilon$ of the gates may be adversarially replaced with any gate g for which $g(0,0) = 0$ and $g(1,1) = 1$. Such a noise is called a *short circuit*, since it can be seen as adversarially connecting one of the inputs to the output, possibly in a way that depends on the input. If gates with fan-in 3 can be used, the noise resilience is $1/6 - \varepsilon$. Furthermore, the compiler is efficient.

Theorem 8.1 ([57])**.** For any $\varepsilon > 0$, there exists an efficient compiler that converts any balanced boolean formula (with fan-in 2 gates) of depth d into formulas E_2 and E_3 that consist of $\vee$ and $\wedge$ gates, and compute the same function as F even if up to e of the $\vee$-gates and up to e of the $\wedge$-gates in any input-to-output path suffer from a short-circuit noise. E_2 contains only gates with fan-in 2, and has depth $4d + 10e$. E_3 contains only gates with fan-in 3, and has depth $2d + 6e$.

The theorem suggests that we can convert F into a formula that is resilient up to a noise fraction of $1/6 - \varepsilon$ with fan-in 3 and up to a noise fraction of $1/10 - \varepsilon$ with fan-in 2. If we set $e_2 = \frac{4d}{\varepsilon}$, then the obtained E_2 can tolerate e_2 noisy gates out of every $4d + 10e_2$ in a single input-to-output path, which amounts to a fraction

$$\frac{e_2}{4d + 10e_2} = \frac{4d/\varepsilon}{4d + 40d/\varepsilon} = \frac{d}{10d + \varepsilon d} \geq \frac{1}{10} - \varepsilon$$

of noisy gates. Similarly, setting $e_3 = 2d/\varepsilon$ guarantees E_3 is resilient to a fraction $1/6 - \varepsilon$ of short-circuit gates. Note that the size of the resilient formula is polynomial in the size of F, that is,

assuming F is balanced,

$$|E_2| \leq 2^{4d+40d/\varepsilon} = (2^d)^{4+40/\varepsilon} = |F|^{4+40/\varepsilon} = \text{poly}_\varepsilon(|F|);$$

similarly $E_3 = \text{poly}_\varepsilon(|F|)$.

The high-level outline of the compiler of Theorem 8.1 is the following:

1. Convert the formula into an interactive protocol (via the Karchmer-Wigderson relation [58]).
2. Compile the interactive protocol to a noise-resilient version (via a rewind-if-error technique).
3. Convert the noise-resilient interactive protocol back into a noise-resilient formula (via Karchmer-Wigderson).

We now give more details about Karchmer-Wigderson games and converting formulas into interactive protocols, and vice versa. For any boolean function, the Karchmer-Wigderson game for f is the following communication task: Alice is given $x \in f^{-1}(0)$ and Bob is given $y \in f^{-1}(1)$. The parties need to find an index i for which $x_i \neq y_i$; note that such an index always exists because $0 = f(x) \neq f(y) = 1$. Karchmer and Wigderson [58] observed that given a formula F (with fan-in 2 $\vee$- and $\wedge$-gates[1]) that computes f, one can obtain an equivalent interactive protocol by the following process: Alice considers $F(x)$ and Bob considers $F(y)$. Start with the output gate of F. If an $\wedge$-gate, Alice sends a single bit $\{L, R\}$ to Bob according to an input of that gate that evaluates to 0. If an $\vee$-gate, Bob sends $\{L, R\}$ to Alice according to the input of that gate that evaluates to 1. The parties then follow to the gate indicated by the message $\{L, R\}$ and repeat, until they reach a leaf that corresponds to a literal for which $x_i \neq y_i$. The correctness can be seen by observing that this process preserves the invariant that the currently considered gate always evaluates to 0 for Alice and to 1 for Bob.

Note that if this protocol is expressed as a tree (i.e., as a pointer-chasing problem, as described in 2.1.3), then it exactly corresponds to the formula F, replacing each $\wedge$-gate with a node that belongs to Alice and any $\vee$-gate with a node that belongs to Bob and replacing the literals of F with output leaves.

After the parties obtain an interactive protocol, they can convert it into a resilient scheme via techniques discussed earlier in this manuscript. It turns out that short-circuit errors translate into substitution errors in the *feedback* model. To see this, consider the Karchmer-Wigderson protocol when a short-circuit noise occurs. In such a case, the new gate to be considered is the one dictated by the short-circuit error rather than by the $\{L, R\}$ symbol sent by the party. But, it is also necessary that *both* parties be aware of this short-circuit noise so that they can agree on the next gate to consider. In other words, they always have a consensus regarding where they stand in the protocol tree. Such a consensus is equivalent to the case of channels with feedback (recall the discussion in §6.1). Indeed, the coding scheme used by [57] to make the interactive protocol resilient to noise "follows"[2] the same ideas as Algorithm 7 (specifically, the variant used for Theorem 6.5) and obtains noise resilience of $1/3-\varepsilon$ and $1/5-\varepsilon$, respectively, for the ternary and binary protocols that correspond to the (naïve) bounds for the feedback channel [27], as discussed in §6.1. Translating

[1] Without loss of generality, we consider only formulas that do not contain $\neg$-gates, since by DeMorgen's rule we can assume all the negations are at the variables.

[2] We note that [57] chronologically precedes Algorithm 7.

the protocol back to a formula results in a loss of another factor of 2 in the maximal resilience, which leads to the bounds of 1/6 and 1/10 stated earlier. Using the more sophisticated machinery of Theorem 6.5, (i.e., Algorithm 8), it is possible to obtain a formula with fan-in 2 that is resilient to a fraction $1/6 - \varepsilon$ of faulty gates.

In order to show the translation from a resilient protocol into a resilient formula, we need the set up some notation. Let P be an interactive protocol, and let V^A and V^B be the set of Alice's nodes and Bob nodes, respectively, in the protocol tree. We define a noise pattern $e = (e^A, e^B)$, with $e^A \in (\Sigma \cup \{\perp\})^{V^A}$ being the noise on Alice's nodes and $e^B \in (\Sigma \cup \{\perp\})^{V^B}$, the noise on Bob's nodes. The error pattern e tells, for each node $v \in V$ in the protocol tree, if the transmission is corrupted or not: if the transmission of node v is corrupted, the pattern $e_v = \sigma$ tells which symbol $\sigma \in \Sigma$ is received by the other side; otherwise $e_v = \perp$ denotes an uncorrupted transmission. Similar to the standard Karchmer-Wigderson conversion, each node in the protocol tree will be replaced with a gate. The noise pattern e on the formula denotes whether the gate at node v is short-circuited or not: if $e_v = \sigma$, it is short-circuited to its σ-input, and if $e_v = \perp$, the gate behaves correctly.

Translating a resilient protocol back into a resilient formula is performed via the following lemma.

Lemma 8.2 ([57]). Let f be a boolean function, and let P be an interactive protocol (assuming noiseless feedback) that correctly solves the Karchmer-Wigderson game for f on any input and noise pattern from $T \times U$, where $T \subseteq f^{-1}(0) \times (\Sigma \cup \{\perp\})^{V^A}$ and $U \subseteq f^{-1}(1) \times (\Sigma \cup \{\perp\})^{V^B}$. Furthermore, assume that each node in the protocol tree of P is reachable by some element in $T \times U$. Then the formula F obtained from the protocol tree of P by replacing every time Alice speaks with an $\wedge$-gate, every time Bob speaks with an $\vee$-gate, and every leaf with a literal correctly computes f for any input and noise in $T \cup U$. That is, for any $(x, a) \in T$, $F(x) = 0$ assuming the noise pattern $e = (a, e^b)$ with any $e^b \in (\Sigma \cup \{\perp\})^{V^B}$. Similarly, for any $(y, b) \in U$, it holds that $F(y) = 1$, assuming the noise $e = (e^A, b)$ for any $e^A \in (\Sigma \cup \{\perp\})^{V^A}$.

Proof. We prove a simpler variant of the lemma, namely, that given some $(x, a) \in T$, it holds that $F(x) = 0$, assuming the noise pattern $e = (a, \perp^{V^B})$. The lemma will then follow, since short-circuiting an $\vee$-gate can turn its output only from 1 to 0, but not in the other direction. The proof for $(y, b) \in U$ is similar.

The proof is done by induction on the depth d of the protocol tree. If $d = 0$, then the protocol simply outputs without communicating. Since the protocol is correct (from Alice's point of view), on each input $(x, \cdot) \in T$, it outputs an index i, so that for any $(y, \cdot) \in U$, $x_i \neq y_i$. Since Alice does not know y, her output i must be independent of y. Similarly, Bob's output is independent of x. Since the protocol is correct, both parties output the same value i, which means that i does not depend on the specific x, y. This implies that either $F(z) = z_i$ or $F(z) = \overline{z_i}$ satisfies the claim.

For the case of $d > 0$, consider the root of the protocol tree (i.e., the first round of the protocol). Assume without loss of generality that Alice owns the root node, so she sends a symbol at the first round. For each $\sigma \in \Sigma$, let T_σ be the set of $(x, a) \in T$ for which Bob receives σ in the corresponding round of the protocol. Let $\Sigma' \subseteq \Sigma$ be the set of σ for which $T_\sigma \neq \emptyset$.

Let $F(z)$ be the formula obtained by converting the protocol tree into a formula, as stated in the lemma. The output of $F(z)$ is an $\wedge$-gate; we let $F_\sigma(z)$ be the subformula connected to the σ-input of the output gate. Since every node in the protocol tree is reachable, the root gate has an

input only for σ for which $T_\sigma \neq \emptyset$, that is,

$$F(z) = \bigwedge_{\sigma \in \Sigma'} F_\sigma(z).$$

For any $\sigma \in \Sigma'$, the induction hypothesis guarantees that $F_\sigma(z)$ correctly computes f for any $T_\sigma \times U$.

Given $(x, a) \in T$, we want to show that $F(x) = 0$ assuming the noise pattern $e = (a, \perp^{V^B})$. Assume $(x, a) \in T_1$, that is, either Alice sends 1 and $e_{root} = \perp$ or $e_{root} = 1$. The induction hypothesis asserts that $F_1(x) = 0$, assuming the noise induced by $(a, \perp^{V^B})$; therefore, $F(x) = 0$ either in the case where $e_{root} = \perp$ or in the case that the first input (i.e., $F_1(x)$) is short-circuited to the output.

Given $(y, b) \in U$, we want to show that $F(y) = 1$, assuming the noise pattern $e = (\perp^{V^A}, b)$. The induction hypothesis asserts that for all $\sigma \in \Sigma'$, it holds that $F_\sigma(y) = 1$, assuming the noise induced by $(\perp^{V^A}, b)$; thus $F(y) = 1$ regardless of e_{root}. □

8.2 Noise-resilient private protocols (do not exist)

"Can noise-resilient protocols be also private?" This question was asked by Chung, Pass, and Telang [22] as well as by Gelles, Sahai, and Wadia [44], trying to employ coding techniques upon distributed protocols that perform secure computation. The somewhat surprising answer to this question is that privacy and noise resilience are contradictory goals that damage each other. While one can always make a private (noiseless) protocol π_0 resilient by the naive approach of encoding each message separately (which has a vanishing rate, and is resilient to at most $O(1/n_0)$ adversarial noise), there is no hope of getting a private protocol that is resilient to more than a fraction of $O(1/n_0)$ corruptions or, in particular, a constant fraction of noise.

Theorem 8.3 ([22, 44]). For any $\varepsilon > 0$, there exists a function f and a private interactive protocol π_0 that computes f, assuming noiseless channels; however, no protocol π for f is both private and resilient to a fraction ε of adversarial noise.

There are several ways to define what "privacy" actually means. Chung et al. [22] require the resilient protocol π to preserve exactly the same information release as in π_0. That is, if π_0 transmits some information b_0 and only then transmits b_1, so should π. Gelles et al. [44] require only that π leaks the same information that π_0 releases overall (e.g., regardless of the order in which information is released).

Both impossibility proofs [22, 44] are based on the same observation: in order to correct errors, the protocol must be able to *rewind* a faulty suffix of the simulation and perform the same part of the computation again, yet with different (intermediate) inputs. This rewinding, however, is fatal in secure computations, as it leaks what the parties do on two different inputs. As an example, consider a zero-knowledge protocol [49], say for graph isomorphism (Figure 8.1, see also [5, Section 9.4]): Given two input (isomorphic) graphs G_1, G_2, at each iteration the prover randomly samples a graph H isomorphic to the input graphs, and it needs to show an isomorphism to either G_1 or to G_2. Assuming the verifier cannot compute isomorphism between two graphs by itself, H gives the verifier no information about the isomorphism $G_1 \simeq G_2$—the verifier could have generated H by itself, by sampling a random isomorphism. The only information the verifier learns is the fact that such an isomorphism exists (if the prover is successful in the protocol). However, if the protocol is allowed to "rewind", for example, if it allows the verifier to replace $\sigma = 1$ with $\sigma = 2$ at Step 2 of

The Graph-Isomorphism Zero-Knowledge Protocol:
Common input: isomorphic graphs $G_1 = (V, E_1), G_2 = (V, E_2)$; The prover is assumed to know an isomorphism $\pi : V \to V$ such that $G_1 = \pi(G_2)$. Repeat for k iterations:

1. The prover picks a random isomorphism $\varphi : V \to V$ and sends the verifier $H = \varphi(G_1)$.
2. The verifier randomly picks $\sigma \in \{1, 2\}$ and challenges the prover to show that $H \simeq G_\sigma$.
3. If $\sigma = 1$, the prover replies with φ; Otherwise, it replies with $\varphi(\pi(\cdot))$.
4. The verifier receives the isomorphism $\tilde{\pi}$ and checks that $H = \tilde{\pi}(G_\sigma)$.

Figure 8.1: A zero-knowledge protocol for graph isomorphism

the same iteration and get an answer (at Step 3) for both challenges, then it provides the verifier with the complete isomorphism, leaking more information that it should have and thus violating privacy.

The result of Chung et al. [22] extends also to the computationally bounded setting, where the adversarial channel (as well as the parties) are computationally bounded. In this setting, an interesting tradeoff between the rate and the noise resilience is provided.

Theorem 8.4 ([22])**.** Assume the existence of a computational private interactive coding scheme with rate r that is resilient to a fraction ε of noise, then $\varepsilon r = o(1/\log(\kappa))$, where κ is the security parameter.

This tradeoff does not eliminate the possibility of having computationally private protocols that are resilient to some constant fraction of noise. Indeed, assuming one-way functions exist, one can obtain a scheme whose rate is constant (which may depend on the security parameter κ). This is done via exchanging long signature keys and then signing each one of the messages transmitted by the noiseless protocol. Since the noise is computationally bounded, it cannot corrupt a message and generate a valid signature. Then, the corruption will be detected by the receiver (with high probability). This has the effect of turning corruption into erasures which assists the parties in advancing the protocol only in the correct track (see §6.2). In particular, the parties would never (i.e., except with a negligible probability) perform computations on invalid intermediate inputs.

Theorem 8.5 ([22])**.** Assume the existence of one-way functions. Then, for every $\varepsilon > 0$, there exists a private interactive coding scheme with rate $O(1/\kappa\varepsilon)$ that is resilient up to a noise rate of $1/12 - \varepsilon$. If, additionally, a subexponentially-hard one-way function exists, the rate can be improved to $O(1/\text{poly}(\log \kappa))$.

Remark 8.1. The preceding impossibilities hold only for adversarial noise. Somewhat surprisingly, random noise makes it easier to perform secure computations. Indeed, in the presence of random noise, one can obtain an *oblivious transfer* primitive [24, 25], with which the secure computation of any function is possible [72, 48]—even functions that cannot be securely computed over a noiseless channel!

8.3 Noise-resilient interactive proofs

While privacy cannot be preserved by interactive protocols, does the same hold for other possible properties of interactive protocol, such as completeness or soundness? This question was asked by Bishop and Dodis [9], extending the discussion to the class **IP** of polynomial-time interactive proofs [49] (see also [5, Section 8]).

Definition 8.1 (polynomial-time interactive proof system [49])**.** An interactive proof for a language L is an interactive protocol between an all-powerful prover P and a polynomial-time verifier V, which on any input x (given to both parties) satisfies

Completeness: If $x \in L$, then at the end of the protocol between V and P, the verifier accepts x with probability at least $2/3$.

Soundness: If $x \notin L$, then for any (possibly cheating) prover $\tilde{P}$, the verifier V accepts x with probability at most $1/3$.

Not all languages L have a protocol that satisfies the preceding, and the set of languages that have such a protocol whose length is polynomial in $|x|$ is known as the complexity class **IP**.

Given some $L \in \mathbf{IP}$ and given an interactive protocol for L that satisfies the conditions of Definition 8.1, it is clear that employing coding techniques would preserve the completeness (i.e., the correctness) of interactive proofs in the presence of a fraction ε of adversarial noise. Bishop and Dodis prove that the same holds for soundness [9].

First, we augment interactive proofs into a noise-resilient version.

Definition 8.2. A polynomial-time *error-resilient* interactive proof for a language L is an interactive protocol between a prover P and a polynomial-time verifier V, which on any input x (given to both parties) satisfies

ε-Noisy Completeness: If $x \in L$, then at the end of the protocol between V and P, the verifier accepts x with probability at least $2/3$, even if up to a fraction ε of the protocol transmissions were adversarially corrupted.

Soundness: If $x \notin L$, then for any (possibly cheating) prover $\tilde{P}$, the verifier V accepts x with probability at most $1/3$.

Define the *error-resilient IP* (**ERIP**) complexity class as the set of all languages L that have a protocol of polynomial length in $|x|$ satisfying Definition 8.2. It then holds that the noise resilient class is equivalent to the standard class.

Theorem 8.6 ([9])**.**

$$\mathbf{ERIP} = \mathbf{IP}.$$

Note that the soundness requirement in Definition 8.2 is the same as in Definition 8.1. Indeed, since the soundness holds for *any* cheating adversary $\tilde{P}$, we can merge the role of the adversarial channel with the role of the cheating prover. The difficulty stems from making the completeness more lenient, as this can be used by the cheating prover to violate the soundness.

Quite informally, the high-level ideas behind the proof of Theorem 8.6 are as follows. We begin by using the framework of Shamir [78] for showing that $\mathbf{IP} = \mathbf{PSPACE}$. In this framework, the

interactive protocol between P and V can be formulated as k iterations of *challenge-response*, where at each iteration the verifier randomly selects a challenge c (possibly depending on the communication so far), and sends it to the prover. The prover needs to answer a response r_c that fits that specific challenge. If $x \in L$, the prover can come up with a "correct" response r_c for any challenge c sent by the verifier. However when $x \notin L$, then for a large fraction of the challenges, there is no correct response; the prover successfully fools the verifier into accepting only in the case where the challenge that the verifier sends comes from the small set of "bad" challenges that do have a valid response; however, this happens with small probability.

It then follows that even if we allow noise and allow the protocol to "rewind" some bounded number of steps back, the probability that the verifier picks a "bad" challenge anywhere during the protocol is small. This probability can be further reduced via parallel repetitions (i.e., performing k-copies of the same protocol with different challenges and accepting only each of the copies is an accepting instance). The downside of this repetition is an increase in the communication of the protocol. The obtained protocol has a vanishing rate of $O(1/r_0)$, where r_0 is the number of rounds in the original (noiseless) protocol [9].

8.4 Noise-resilient perpetual (one-way) communication

In this section we describe several applications of tree codes and related coding techniques applied in the setting of data-stream communication.

Definition 8.3 (data stream)**.** A data stream S over an alphabet Γ is an (infinite) string $S = s_1 s_2 s_3 \cdots$, where each $s_i \in \Sigma$ arrives at time i (and is unknown beforehand)

Assume Alice generates a stream S that she wishes to transmit to Bob over some channel. That is, at every given time, Bob wishes to know the stream generated by Alice up to that time. If the communication is noiseless, then at every timestep Alice would simply send the symbol s_i that was just generated at that time. The question is how to handle the datastream's transmission over an adversarially noisy channel.

A naive encoding for this setting would partition the stream into blocks of size B and encode each block separately using a good error-correcting code. However, it is easy to see that such an approach cannot tolerate B adversarial corruptions, since these can completely corrupt the encoding of an entire block and would permanently prevent Bob from learning the corresponding datastream symbols that were contained in the corrupted block. A desired goal is that no matter what the noise does, Bob should *eventually* learn any s_i generated by Alice. Specifically, we would like a coding scheme that is resilient to a constant fraction of noise and has a constant rate.

Since the stream (and, thus, the encoding) is infinite, we need to be more careful when defining the noise rate and the rate of the code. Similar to the case of protocols with varying length (§4), we can define a relative version of these quantities, measured up to some time n. We will assume the encoding has a fixed (nonvarying) rate. Then, the rate is simply $\frac{\log|\Sigma|}{M \log|\Gamma|}$, where Γ is the data stream's alphabet, Σ is the channel's alphabet, and M is the number of symbols communicated at every time step. The fraction of noise *up to some time* n is the number of corruptions performed through time n divided by Mn, the number of transmissions up to time n.

One issue that appears in this setting is that at time n we cannot expect the receiver to be able to decode the entire stream up to time n. Indeed, assume that all the transmissions between

times $(1-c)n$ and n are corrupted for some constant $c > 0$; the noise fraction in this case is c. It is easy to see that the receiver cannot learn $s_{(1-c)n} \cdots s_n$: before time $(1-c)n$ these elements were still unknown (at the sender's side), while after that time all the transmissions were corrupted.

It is not very surprising that an (infinite) tree code provides a good coding scheme in this setting. A simple encoding of the stream S through a tree code allows the receiver to decode a prefix of S of length at least approximately εn at any time n for which the fraction of noise is at most $1/2 - \varepsilon$.

Theorem 8.7 ([35])**.** For any constants $\varepsilon > 0$ and $\delta > 0$, there exists a constant-rate error-correction scheme for data stream $S = s_1, s_2, \ldots$ such that at any given time n the receiver outputs a string $s'_1, s'_2, \ldots, s'_n$, and if the noise rate until time n is at most $1/2 - \varepsilon$, then

$$s'_1 s'_2 \cdots s'_{(2\varepsilon-\delta)n} = s_1 s_2 \cdots s_{(2\varepsilon-\delta)n}.$$

Proof. Let $\mathcal{T}$ be an infinite $|\Gamma|$-ary tree code with alphabet Σ and distance $\alpha \geq \frac{1-2\varepsilon}{1-2\varepsilon+\delta}$, the existence of which is given by Theorem 2.2. Alice encodes S by simply encoding each s_i using the tree code. That is, at the i-th time step she transmits the last symbol of $\mathsf{TCenc}(s_1 \cdots s_i)$. Using Theorem 2.2, the rate of the scheme is given by

$$\frac{\log|\Gamma|}{\log|\Sigma|} = \frac{\log|\Gamma|}{O(\frac{1}{1-\alpha})\log|\Gamma|} = O\left(\frac{1}{1-\alpha}\right) = O_{\varepsilon,\delta}(1).$$

For a specific time n, consider a string $s' \in \{0,1\}^n$ that agrees with $s_1, \ldots, s_n$ on a prefix of length less than $(2\varepsilon - \delta)n$. Due to the tree distance property, the Hamming distance between $\mathsf{TCenc}(s'_1 \cdots s'_n)$ and $\mathsf{TCenc}(s_1 \cdots s_n)$ is at least $\alpha(1-2\varepsilon+\delta)n$, which means the decoding will output $s'_1 \cdots s'_n$ instead of $s_1 \cdots s_n$ only if there were at least $\lceil \alpha(1-2\varepsilon+\delta)n/2 \rceil$ corruptions. However, the corruption rate in this case is

$$\frac{\lceil \alpha(1-2\varepsilon+\delta)n/2 \rceil}{n} \geq \frac{\frac{1-2\varepsilon}{1-2\varepsilon+\delta}(1-2\varepsilon+\delta)}{2} \geq \frac{1}{2} - \varepsilon. \qquad \square$$

One can verify that with adversarial noise rate of $1/2$, no communication can be carried out, and so the preceding relation between the noise rate and the decodable prefix is tight. If we relax the setting, for example, if we allow the parties to possess a secret random string (unknown to the adversary), then an (efficient) coding scheme exists that for any time n in which the noise fraction is at most $1 - \varepsilon$ decodes a prefix of length at least $\approx \varepsilon n$ [35].

Ostrovsky, Rabani, and Schulman [68] considered another interesting relaxation of the setting, which is inspired by automatic control. Here, the sender's stream is seen as its internal state, which it wishes to communicate to the receiver. Given a certain state s_i at time i, the sender shifts at time $i+1$ to another state according to some fixed transition function that depends only on s_i and i (but not on the history). Ostrovsky et al. define a family of codes named *trajectory codes* in which the encoding of s_i may depend only on i and s_i (and does not depend on $s_1 \cdots s_{i-1}$). In trajectory codes, the Hamming distance between the encodings of two divergent paths is proportional to the Hamming distance of the two divergent paths (rather than to the *length* of the divergent paths, as in the case of tree codes).

Trajectory codes extend the notion of tree codes. Indeed, when the underlying state space (i.e., the possible states and their connections, given by the transition function) is a tree, the obtained

code is equivalent to a tree code. However, in other special cases, for example, when the set of states grow polynomially with time (rather than exponentially, as in the case of a tree), better codes can be found. In [68], asymptotically good trajectory codes (that is, codes with a constant rate and any distance less than 1) are shown to exist for any possible transition function [68]. Moreover, *efficient* constructions are provided for the interesting case where the set of states and the transition function correspond to a d-dimensional grid.

Open Questions for Section 8

1. Is it possible to use the techniques of §8.1 to obtain noise-resilient *circuits* (i.e., with an arbitrary fan-out) of polynomial size in the noiseless circuit?
2. Can the communication-efficient schemes of §6 be used to minimize the size of a resilient circuit?
3. When sending a message over a unidirectional setting, the process can be separated into two *independent* parts: first compressing the message (using some source-coding technique) and then encoding the compressed message (using a some channel-coding technique) [80]. Does the same source-channel separation hold in the interactive case? That is, do general-purpose coding schemes perform as well as coding schemes tailored for a certain application?
4. What other applications can benefit from interactive coding techniques?

A

Addendum

The first version of this manuscript was completed in mid-2015 and included works that were published by that time. In this appendix we briefly mention recent works that appeared after the completion of the first version.

A.1 The Hunt for Efficient Constructions (Section 3):

Gelles, Haeupler, Kol, Ron-Zewi, and Wigderson [39] consider efficient *deterministic* interactive coding schemes with high rate of $1 - O(\sqrt{H(\varepsilon)})$. The approach of [39], borrowed from the uni-directional regime, is the approach of *concatenation* [33]. The code is composed of two layers: the first layer (the "inner code") may be inefficient, but has optimal parameters, while the second layer (the "outer code") needs to be efficient but may have sub-optimal parameters. Several previous works (e.g, [11] as well as [59, 51] described in §3) indeed follow this high-level idea of splitting the scheme into small chunks, simulating each chunk with a (possibly inefficient) interactive code serving as an inner code. The outer code, in this case, needs to "synchronize" the simulation in between blocks.

The coding scheme of Gelles et al. [39] follows this path explicitly, and shows a deterministic and efficient construction of a linear *systematic* tree code, that can be used as an outer code.

Definition A.1 (linear systematic tree codes [39])**.** A 2^k-ary tree code $\mathcal{T}$ over alphabet $\Sigma = \{0,1\}^n$ is *systematic* if the following holds: if $\mathsf{TCenc}(x_1, \ldots, x_\ell) = \sigma_1, \ldots, \sigma_\ell$, then for any $i \in [\ell]$, the first k-bits in the description of σ_i equal x_i. The tree is *linear*, if for every ℓ, $\mathsf{TCenc} : \{0,1\}^{k\ell} \to \{0,1\}^{n\ell}$ is a linear map.

In particular, they show how to construct a systematic linear $O(n)$-ary tree code of length n with distance $O(1/\log n)$. Concatenating the above outer code with any coding scheme for BSC channels (possibly inefficient scheme, e.g., Algorithm 1) that serves as an inner code gives an efficient simulation with success probability $1 - 2^{-\Omega(n_0/\log n_0)}$ over BSC_ε.

Theorem A.1 ([39]). For any $\varepsilon < 1/2$ there exists an efficient and deterministic coding scheme with rate $1 - O(\sqrt{H(\varepsilon)})$, that simulates any (alternating) π_0 over a BSC_ε with probability $1 - 2^{-\Omega(n_0/\log n_0)}$.

A.2 Communication-Efficient Coding Schemes (Section 5)

Haeupler and Velingker [52] consider the rate of two-party interactive coding schemes when up to a fraction ε of the bits are corrupted by the channel. Previous work (Kol and Raz [59] and Haeupler [51] described in §5) demonstrated that the maximal rate for interactive coding in this setting is $\approx 1 - O(\sqrt{H(\varepsilon)})$, while coding in the one-way setting can be performed with rates as high as $1 - O(H(\varepsilon))$ [53].

Haeupler and Velingker find that this quadratic gap can be bridged for a large class of interactive protocols where the parties' order of speaking is not alternating. Instead, the parties communicate messages of length $\Omega(\text{poly}(1/\varepsilon))$ bits, on average. The noise model allows corrupting up to a fraction ε of the communication. However, the channel is assumed to be *oblivious*, i.e., the noise pattern is fixed and independent of the communicated transcript.

Theorem A.2 ([52]). Any protocol π_0 in which parties send messages of average length $\Omega(\text{poly}(1/\varepsilon))$ can be simulated over an oblivious noisy channel that corrupts up to a fraction ε of the transmissions, with a rate of $1 - \Theta(H(\varepsilon))$ and a success probability of $1 - 2^{-\Omega(n_0\varepsilon^6)}$.

A.3 Coding Schemes over Different Noisy Channels (Section 6):

Sherstov and Wu [81] consider two-party interactive coding over channels with insertions and deletions. Building on top of the scheme of Braverman et al. [17], they show a coding scheme that is resilient to a fraction $1/4$ of insertions and deletions (in the setting of [17]). Recall that a noise resilience of $1/4$ is maximal even for a simpler noise, namely, one that only flips bits. This makes the scheme of Sherstov and Wu optimal in this sense as well.

Theorem A.3 ([81]). For any $\varepsilon > 0$, there exists a coding scheme with a constant rate that simulates any protocol π_0 over a channel with insertions and deletions, has a constant rate, and is resilient to a fraction $1/4 - \varepsilon$ of corruptions.

This, in fact, completely solves Open Question 4 described in §6.

A.4 Multiparty Interactive Communication (Section 7):

Braverman, Efremenko, Gelles, and Haupler [16] consider multiparty interactive coding over BSC_ε channels in the synchronous setting. Recall that in this setting the coding scheme of Rajagopalan and Schulman [73] (Algorithm 11) achieves a rate of $\Theta(1/\log d)$, where d is the maximal degree in the network $G = (V, E)$ with $m = |V|$ and $l = |E|$. Braverman et al. show that this rate is essentially optimal. specifically, they show a communication task over a star network, for which any coding scheme has a rate of at most $O(1/\log m)$.

Gelles and Kalai [40] revisit the question of rate in the synchronous multiparty case, and ask what happens if we don't require every party to communicate with all its neighbors at every round. Based on the techniques in [16] they show a communication task over a cycle network for which any

coding scheme has a rate of at most $O(1/\log m)$. Note that in a cycle network any party has exactly two neighbors, hence $d = 2$ and Algorithm 11 has a constant rate. The reason for this discrepancy is simple: the fact that in the setting of [73] ever party speaks with all its neighbors at every round implies an inherent increase in the communication by a factor of $O(m)$ per round, which allows the parties to overcome errors. However, this increase in the communication exists both in the noiseless protocol and in the coding scheme, so it cancels out in the rate. Gelles and Kalai suggest that the setting of [73] actually focuses on the round complexity rather on the communication complexity of the coding scheme.

Censor-Hillel, Gelles, and Haeupler [20] consider multiparty coding schemes in the asynchronous setting, where the topology of the network G is not known in advance, that is, each party only knows its own neighbors. By combining techniques from the literature of distributed computing and interactive coding, they devise a distributed asynchronous coding scheme that is resilient to the maximal fraction $O(1/m)$ of noise and demonstrates a rate of $O(1/m \log^2 m)$.

Theorem A.4 ([20])**.** For any graph G, and any noiseless asynchronous protocol π_0 that sends n_0 bits over G, there exists a deterministic, asynchronous coding scheme that simulates π_0 over the noisy network G. The scheme communicates $O(n_0 \cdot m \log m)$ messages of size $O(\log m)$ each, and is resilient to a fraction $\Theta(1/m)$ of adversarial noise.

Efremenko, Kol, and Sexana [29] consider multiparty interactive coding in a setting where the parties share a noisy *broadcast channel* [30]: when a party broadcasts the bit b, all the other parties receive an independent noisy version of b. I.e., the i-th party receives $b_i = b \oplus e_i$ where $e_i = 1$ with probability ε and $e_i = 0$ with probability $1 - \varepsilon$, independently of all $\{e_j\}$ for $j \neq i$.

This setting was previously examined by Gallager [37], who showed a coding scheme in which each party begins with an input bit x_i and at the end of which all parties know all the inputs $\{x_i\}$ with high probability. Gallager's scheme takes $O(m \log \log m)$ broadcasts, i.e., it has a rate of $\Omega(1/\log \log m)$. Later, Goyal, Kindler, and Saks [50] showed a matching upper bound on the rate, for the same task.

Efremenko et al. [29] slightly change the setting in the following sense: the protocol works in rounds in each of which any party is allowed to broadcast one bit. However, if two or more parties broadcast at the same round their transmissions collide. In this case the, each one of the other parties hears an adversarially chosen bit. (Compare to the adaptive setting in §4.)

In this setting, they show an interactive coding scheme with a constant rate.

Theorem A.5 ([29])**.** There exists a randomized adaptive coding scheme for noisy broadcast channels that simulate any (nonadaptive) noiseless protocol, takes $O(n_0)$ broadcast transmissions and errs with sub-constant probability in n_0.

B

Summary of Known Schemes

In this appendix we summarize the coding schemes surveyed in this manuscript. **Table B.1** summarizes coding schemes in the random noise setting. **Table B.2** summarizes coding scheme in the adversarial noise model. **Table B.3** summarizes adaptive coding schemes. Finally, **Table B.4** summarizes coding schemes in the multiparty setting. The parameter n denotes the length of the coding scheme.

Table B.1: Coding schemes over channels with noise probability ε

Scheme	Rate	Alphabet	Success Probability	Efficient?	Deterministic?	Notes
[Sch92]	$\Theta(1)$	$O_\varepsilon(1)$	$1-2^{-\Omega_\varepsilon(n/\log^c(n))}$	$\surd$		
[Sch96]	$\Theta(1)$	$O_\varepsilon(1)$	$1-2^{-\Omega_\varepsilon(n)}$		$\surd$	
[Sch03]	$\Theta(1)$	$O_\varepsilon(1)$	$1-1/n^c$	$\surd$	$\surd$	
[Pac06]	$\Theta(\log^{-\frac{1}{2}}(1/p))$	$O_\varepsilon(\sqrt{\log(1/p)})$	$1-p$	$\surd$		
[Bra12]	$\Theta(1)$	$O_\varepsilon(1)$	$1-2^{-\Omega_\varepsilon((\log n)^c)}$	$\surd$	$\surd$	
[GMS14]	$\Theta(1)$	$O_\varepsilon(1)$	$1-2^{-\Omega_\varepsilon(n)}$	$\surd$		
[KR13]	$1-O(\sqrt{\varepsilon\log 1/\varepsilon})$	binary	$1-2^{-\Omega_\varepsilon(n^c)}$	$\surd$		(3)
[Hau14]	$1-O(\sqrt{\varepsilon})$	binary	$1-2^{-\Omega_\varepsilon(n)}$	$\surd$		
	$1-O(\sqrt{\varepsilon\log\log 1/\varepsilon})$	binary	$1-2^{-\Omega_\varepsilon(n)}$	$\surd$		(1)
[GHK+16]	$1-O(\sqrt{\varepsilon\log 1/\varepsilon})$	binary	$1-2^{-\Omega_\varepsilon(n/\log n)}$	$\surd$	$\surd$	
[Pan13]	$1-O(\sqrt{\varepsilon})$	binary	$1-2^{-\Omega_\varepsilon(n)}$	$\surd$	$\surd$	(1),(2)
[GH15]	$1-O(\varepsilon\log 1/\varepsilon)$	binary	$1-2^{-\Omega_\varepsilon(n)}$	$\surd$		(1),(2)
[HV17]	$1-O(\varepsilon\log 1/\varepsilon)$	binary	$1-2^{-\Omega_\varepsilon(n)}$	$\surd$		(4)

Notes: (1) The same result applies also in the case of a fraction ε of adversarial noise.

(2) Over channels with feedback and erasure channels.

(3) The scheme assumes a shared random string; however, this assumption can be removed using standard techniques. See Remark 5.1.

(4) Assuming the noiseless protocol communicates messages of length $\Omega(\text{poly}(1/\varepsilon))$ on average.

Table B.2: Coding schemes over channels with a fraction $M - \varepsilon$ of adversarial noise

Scheme	Maximal noise M	Alphabet	Success Probability	Efficient?	Deterministic?	Setting
[Sch96]	1/240	binary	1		√	
[BR14]	1/4	$O_\varepsilon(1)$	1		√	
	1/8	binary	1		√	
[BE14]	1/4 (*)	$O_\varepsilon(1)$	1		√	
[BKN14]	1/16	$O_\varepsilon(1)$	$1 - 2^{-\Omega(n)}$	√		
	1/32	binary	$1 - 2^{-\Omega(n)}$	√		
[GH14]	1/4	$O_\varepsilon(1)$	$1 - 2^{-\Omega(n)}$	√		
[FGOS15]	1/2	$O_\varepsilon(1)$	$1 - 2^{-\Omega(n)}$			shared randomness
	1/2	$O_\varepsilon(1)$	1		√	erasure channel
[EGH16]	1/4	3	1	√	√	feedback, fixed order
	1/6	binary	1	√	√	feedback, fixed order
	1/3	binary	1	√	√	feedback, varying order
	1/2	4	1	√	√	erasure channel
	1/3	binary	1	√	√	erasure channel
[BGMO16]	1/18	$O_\varepsilon(1)$	1		√	channel with insertions and deletions
[SW17]	1/4	$O_\varepsilon(1)$	1		√	channel with insertions and deletions
[BNT+14]	1/2	$O_\varepsilon(1)$	$1 - 2^{-\Omega(n)}$			quantum protocols, shared entanglement

Notes: (*) The noise can be refined to the specific amount affecting each direction.

Table B.3: Adaptive coding schemes over channels with a fraction $M - \varepsilon$ of adversarial noise

Scheme	Maximal noise M	Rate	Alphabet	Success Probability	Setting
[GHS14]	2/7	$\Theta_\varepsilon(1)$	$O_\varepsilon(1)$	$1 - 2^{-\Omega(n)}$	
	2/3	$\Theta_\varepsilon(1)$	$O_\varepsilon(1)$	$1 - 2^{-\Omega(n)}$	shared randomness
[AGS16]	1/3	$\exp(-n)$	$O_\varepsilon(1)$	1	adaptive length only
	2/3	$\exp(-n)$	binary	1	
	1/2	$\Theta_\varepsilon(1)$	binary	1	
	1	$\Theta_\varepsilon(1)$	binary	$1 - 2^{-\Omega(n)}$	shared randomness
	1	$\Theta_\varepsilon(1)$	binary	1	erasure channel

Table B.4: Multiparty coding schemes

Scheme	Topology	Noise	Rate	Message Passing	Success Probability	Notes
[RS94]	any	BSC_ε	$\Omega_\varepsilon(\frac{1}{\log(d+1)})$	synch.	$1-\lvert V\rvert 2^{-\Omega_\varepsilon(n)}$	d = maximal degree
[ABE+16]	d-regular	BSC_ε	$\Omega_\varepsilon(\frac{1}{t^3 \log t})$	synch.	$1-(t\lvert V\rvert^2 2^{-\Omega(d^\alpha)})^{\Omega(n)}$	t = mixing time; $\alpha < 1$
[EKS17]	broadcast	BSC_ε	$\Theta_\varepsilon(1)$	synch.	$1-n^{-\Omega(1)}$	adaptive
[HS16]	any	$\Theta(\frac{1}{\lvert E\rvert})$	$\Theta(1)$	synch.	1	
	any	$\Theta(\frac{1}{\lvert V\rvert})$	$\Theta(\frac{\lvert V\rvert}{\lvert E\rvert \log \lvert V\rvert})$	synch.	1	
[JKL15]	star (*)	$\Theta(\frac{1}{\lvert V\rvert})$	$\Theta(1)$	asynch.	1	
[LV15]	complete	$\Theta(\frac{1}{\lvert V\rvert})$	$\Theta(1)$	asynch.	1	balanced communication
[CGH17]	any	$\Theta(\frac{1}{\lvert V\rvert})$	$\Omega(\frac{1}{\lvert V\rvert \log^2 \lvert V\rvert})$	asynch.	1	unknown topology (distributed)

Notes: Performed over a network $G = (V, E)$.
(*) or any G that contains a star as a subgraph

References

[1] Shweta Agrawal, Ran Gelles, and Amit Sahai. Adaptive protocols for interactive communication. In *2016 IEEE International Symposium on Information Theory (ISIT)*, pages 595–599, July 2016. See also longer version in arXiv:1312.4182 (cs.DS).

[2] Noga Alon, Mark Braverman, Klim Efremenko, Ran Gelles, and Bernhard Haeupler. Reliable communication over highly connected noisy networks. In *Proceedings of the 2016 ACM Symposium on Principles of Distributed Computing*, PODC '16, pages 165–173, New York, NY, USA, 2016. ACM.

[3] Noga Alon, Oded Goldreich, Johan Håstad, and René Peralta. Simple constructions of almost k-wise independent random variables. *Random Structures & Algorithms*, 3(3):289–304, 1992.

[4] Noga Alon and Joel H. Spencer. *The Probabilistic Method*. Wiley-Interscience Series in Discrete Mathematics and Optimization. John Wiley & Sons, Inc., Hoboken, NJ, 3rd edition, 2008.

[5] Sanjeev Arora and Boaz Barak. *Computational Complexity: A Modern Approach*. Cambridge University Press, 2009.

[6] Charles H. Bennett, Gilles Brassard, Claude Crépeau, Richard Jozsa, Asher Peres, and William K. Wootters. Teleporting an unknown quantum state via dual classical and Einstein-Podolsky-Rosen channels. *Phys. Rev. Lett.*, 70:1895–1899, March 1993.

[7] Elwyn R. Berlekamp. *Block Coding with Noiseless Feedback*. PhD thesis, Massachusetts Institute of Technology, 1964.

[8] M. R. Best, A. E. Brouwer, F. Jessie MacWilliams, Andrew M. Odlyzko, and Niel J. A. Sloane. Bounds for binary codes of length less than 25. *Information Theory, IEEE Transactions on*, 24(1):81–93, January 1978.

[9] Allison Bishop and Yevgeniy Dodis. Interactive coding for interactive proofs. In Eyal Kushilevitz and Tal Malkin, editors, *Theory of Cryptography: 13th International Conference, TCC 2016-A, Tel Aviv, Israel, January 10-13, 2016, Proceedings, Part II*, pages 352–366, Berlin, Heidelberg, 2016. Springer Berlin Heidelberg.

[10] Zvika Brakerski and Yael T. Kalai. Efficient interactive coding against adversarial noise. *Foundations of Computer Science (FOCS), IEEE Annual Symposium on*, pages 160–166, 2012.

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

[12] Zvika Brakerski and Moni Naor. Fast algorithms for interactive coding. In *Proceedings of the 24th Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '13, pages 443–456, 2013.

[13] Gilles Brassard, Ashwin Nayak, Alain Tapp, Dave Touchette, and Falk Unger. Noisy interactive quantum communication. In *Foundations of Computer Science (FOCS), 2014 IEEE 55th Annual Symposium on*, FOCS, pages 296–305, Oct 2014.

[14] Mark Braverman. Towards deterministic tree code constructions. In *Proceedings of the 3rd Innovations in Theoretical Computer Science Conference*, ITCS '12, pages 161–167. ACM, 2012.

[15] Mark Braverman and Klim Efremenko. List and unique coding for interactive communication in the presence of adversarial noise. In *Foundations of Computer Science (FOCS), IEEE 55th Annual Symposium on*, pages 236–245, 2014.

[16] Mark Braverman, Klim Efremenko, Ran Gelles, and Bernhard Haeupler. Constant-rate coding for multiparty interactive communication is impossible. In *Proceedings of the Forty-eighth Annual ACM Symposium on Theory of Computing*, STOC '16, pages 999–1010, New York, NY, USA, 2016. ACM.

[17] Mark Braverman, Ran Gelles, Jieming Mao, and Rafail Ostrovsky. Coding for Interactive Communication Correcting Insertions and Deletions. In *43rd International Colloquium on Automata, Languages, and Programming (ICALP 2016)*, volume 55 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 61:1–61:14, Dagstuhl, Germany, 2016.

[18] Mark Braverman and Anup Rao. Towards coding for maximum errors in interactive communication. In *STOC '11: Proceedings of the 43rd Annual ACM Symposium on Theory of Computing*, pages 159–166, 2011.

[19] Mark Braverman and Anup Rao. Toward coding for maximum errors in interactive communication. *Information Theory, IEEE Transactions on*, 60(11):7248–7255, November 2014.

[20] Keren Censor-Hillel, Ran Gelles, and Bernhard Haeupler. Making asynchronous distributed computations robust to noise. Preprint arXiv:1702.07403 (cs.DS), 2017.

[21] What is the average length of a game of chess?, 2013. Chess StackExchange. [Online:] `https://chess.stackexchange.com/questions/2506/what-is-the-average-length-of-a-game-of-chess`.

[22] Kai-Min Chung, Rafael Pass, and Sidharth Telang. Knowledge-preserving interactive coding. In *Proceedings of the 54th annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 449–458, 2013.

[23] Richard Cleve and Harry Buhrman. Substituting quantum entanglement for communication. *Phys. Rev. A*, 56:1201–1204, August 1997.

[24] Claude Crépeau and Joe Kilian. Achieving oblivious transfer using weakened security assumptions. In *Foundations of Computer Science, 29th Annual Symposium on*, pages 42–52, 1988.

[25] Ivan Damgård, Serge Fehr, Kirill Morozov, and Louis Salvail. Unfair noisy channels and oblivious transfer. In Moni Naor, editor, *Theory of Cryptography*, volume 2951 of *Lecture Notes in Computer Science*, pages 355–373. Springer Berlin Heidelberg, 2004.

[26] Marcel Kenji de Carli Silva, Nicholas J. A. Harvey, and Cristiane M. Sato. Sparse sums of positive semidefinite matrices. Preprint arXiv:1107.0088, 2011.

[27] Klim Efremenko, Ran Gelles, and Bernhard Haeupler. Maximal noise in interactive communication over erasure channels and channels with feedback. In *Proceedings of the 2015 Conference on Innovations in Theoretical Computer Science*, ITCS '15, pages 11–20. ACM, 2015.

[28] Klim Efremenko, Ran Gelles, and Bernhard Haeupler. Maximal noise in interactive communication over erasure channels and channels with feedback. *IEEE Transactions on Information Theory*, 62(8):4575–4588, Aug 2016.

[29] Klim Efremenko, Gillat Kol, and Raghuvansh Saxena. Interactive coding over the noisy broadcast channel. *Electronic Colloquium on Computational Complexity (ECCC)*, 24:93, 2017.

[30] Abbas El Gamal. Open problems presented at the 1984 workshop on specific problems in communication and computation sponsored by bell communication research. In Thomas M. Cover and B. Gopinath, editors, *Open problems in communication and computation*. Springer-Verlag New York, Inc., 1987.

[31] Peter Elias. List decoding for noisy channels. Technical Report 335, Research Laboratory of Electronics, Massachusetts Institute of Technology, 1957. Reprinted from the 1957 IRE WESCON convention record Part 2.

[32] Paul Erdős and László Lovász. Problems and results on 3-chromatic hypergraphs and some related questions. *Infinite and finite sets*, 10:609–627, 1975.

[33] G. David Forney. Concatenated codes. Technical Report 440, Massachusetts Institute of Technology. Research Laboratory of Electronics, 1965.

[34] Matthew Franklin, Ran Gelles, Rafail Ostrovsky, and Leonard J. Schulman. Optimal coding for streaming authentication and interactive communication. In *Advances in Cryptology – CRYPTO 2013*, volume 8043 of *Lecture Notes in Computer Science*, pages 258–276. Springer Berlin Heidelberg, 2013.

[35] Matthew Franklin, Ran Gelles, Rafail Ostrovsky, and Leonard J. Schulman. Optimal coding for streaming authentication and interactive communication. *Information Theory, IEEE Transactions on*, 61(1):133–145, January 2015.

[36] Robert G. Gallager. *Information Theory and Reliable Communication*, volume 2. Springer, 1968.

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

[38] Ran Gelles and Bernhard Haeupler. Capacity of interactive communication over erasure channels and channels with feedback. In *Proceedings of the 26th Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '15, pages 1296–1311, 2015.

[39] Ran Gelles, Bernhard Haeupler, Gillat Kol, Noga Ron-Zewi, and Avi Wigderson. Towards optimal deterministic coding for interactive communication. In *Proceedings of the Twenty-seventh Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '16, pages 1922–1936, Philadelphia, PA, USA, 2016. Society for Industrial and Applied Mathematics.

[40] Ran Gelles and Yael T. Kalai. Constant-rate interactive coding is impossible, even in constant-degree networks. In *Proceedings of the 8th Conference on Innovations in Theoretical Computer Science, ITCS '17*, 2017.

[41] Ran Gelles, Ankur Moitra, and Amit Sahai. Efficient and explicit coding for interactive communication. In *Foundations of Computer Science (FOCS), IEEE 52nd Annual Symposium on*, pages 768–777, 2011.

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

[43] Ran Gelles and Amit Sahai. Potent tree codes and their applications: Coding for interactive communication, revisited. Preprint arXiv:1104.0739 (cs.DS), 2011.

[44] Ran Gelles, Amit Sahai, and Akshay Wadia. Private interactive communication across an adversarial channel. In *Proceedings of the 5th Conference on Innovations in Theoretical Computer Science*, ITCS '14, pages 135–144. ACM, 2014.

[45] Mohsen Ghaffari and Bernhard Haeupler. Optimal error rates for interactive coding II: Efficiency and list decoding. In *Foundations of Computer Science (FOCS), IEEE 55th Annual Symposium on*, pages 394–403, 2014.

[46] Mohsen Ghaffari, Bernhard Haeupler, and Madhu Sudan. Optimal error rates for interactive coding I: Adaptivity and other settings. In *STOC '14: Proceedings of the 46th Annual ACM Symposium on Theory of Computing*, pages 794–803, 2014.

[47] E. N. Gilbert, F. J. MacWilliams, and N. J. A. Sloane. Codes which detect deception. *Bell System Technical Journal*, 53(3):405–424, March 1974.

[48] Oded Goldreich, Silvio Micali, and Avi Wigderson. How to play any mental game. In *Proceedings of the Nineteenth Annual ACM Symposium on Theory of Computing*, STOC '87, pages 218–229. ACM, 1987.

[49] Shafi Goldwasser, Silvio Micali, and Charles Rackoff. The knowledge complexity of interactive proof systems. *SIAM Journal on Computing*, 18(1):186–208, 1989.

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

[51] Bernhard Haeupler. Interactive channel capacity revisited. In *Foundations of Computer Science (FOCS), IEEE 55th Annual Symposium on*, pages 226–235, 2014.

[52] Bernhard Haeupler and Ameya Velingker. Bridging the capacity gap between interactive and one-way communication. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '17, pages 2123–2142. Society for Industrial and Applied Mathematics, 2017.

[53] Richard W. Hamming. Error detecting and error correcting codes. *Bell System technical journal*, 29(2):147–160, April 1950.

[54] Eric Holcomb. So how many chess board positions are there? [online]: `www.nwchess.com/articles/misc/Chess_Board_Positions_article.pdf`.

[55] William M. Hoza and Leonard J. Schulman. The adversarial noise threshold for distributed protocols. In *Proceedings of the Twenty-seventh Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '16, pages 240–258, Philadelphia, PA, USA, 2016. Society for Industrial and Applied Mathematics.

[56] Abhishek Jain, Yael T. Kalai, and Allison B. Lewko. Interactive coding for multiparty protocols. In *Proceedings of the 2015 Conference on Innovations in Theoretical Computer Science*, ITCS '15, pages 1–10. ACM, 2015.

[57] Yael T. Kalai, Allison B. Lewko, and Anup Rao. Formulas resilient to short-circuit errors. In *Foundations of Computer Science (FOCS), 2012 IEEE 53rd Annual Symposium on*, pages 490–499, 2012.

[58] Mauricio Karchmer and Avi Wigderson. Monotone circuits for connectivity require super-logarithmic depth. *SIAM Journal on Discrete Mathematics*, 3(2):255–265, 1990.

[59] Gillat Kol and Ran Raz. Interactive channel capacity. In *STOC '13: Proceedings of the 45th Annual ACM Symposium on Theory of Computing*, pages 715–724, 2013.

[60] Eyal Kushilevitz and Noam Nisan. *Communication Complexity*. Cambridge University Press, 1997.

[61] F.T. Leighton, Bruce M. Maggs, and Satish B. Rao. Packet routing and job-shop scheduling in O(congestion + dilation) steps. *Combinatorica*, 14(2):167–186, June 1994.

[62] Allison Lewko and Ellen Vitercik. Balancing communication for multi-party interactive coding. Preprint arXiv:1503.06381, 2015.

[63] Nathan Linial, Eran London, and Yuri Rabinovich. The geometry of graphs and some of its algorithmic applications. *Combinatorica*, 15(2):215–245, June 1995.

[64] Richard Lipton. A new approach to information theory. In Patrice Enjalbert, Ernst Mayr, and Klaus Wagner, editors, *STACS '94*, volume 775 of *Lecture Notes in Computer Science*, pages 699–708. Springer, 1994.

[65] Ankur Moitra. Efficiently coding for interactive communication. *Electronic Colloquium on Computational Complexity (ECCC)*, 2011. TR11-042.

[66] Cristopher Moore and Leonard J. Schulman. Tree codes and a conjecture on exponential sums. In *Proceedings of the 5th Conference on Innovations in Theoretical Computer Science*, ITCS '14, pages 145–154. ACM, 2014.

[67] Joseph Naor and Moni Naor. Small-bias probability spaces: Efficient constructions and applications. *SIAM Journal on Computing*, 22(4):838–856, 1993.

[68] Rafail Ostrovsky, Yuval Rabani, and Leonard J. Schulman. Error-correcting codes for automatic control. *Information Theory, IEEE Transactions on*, 55(7):2931–2941, July 2009.

[69] Denis Pankratov. On the power of feedback in interactive channels. [Online:] `http://people.cs.uchicago.edu/~pankratov/papers/feedback.pdf`, 2013.

[70] Marcin Peczarski. An improvement of the tree code construction. *Information Processing Letters*, 99(3):92–95, August 2006.

[71] Pavel Pudlák. Linear tree codes and the problem of explicit constructions. *Linear Algebra and its Applications*, 490:124–144, 2016.

[72] Michael O. Rabin. How to exchange secrets with oblivious transfer. IACR Cryptology ePrint Archive, 2005. Originally published as: Technical Report TR-81, Aiken Computation Lab, Harvard University, 1981.

[73] Sridhar Rajagopalan and Leonard J. Schulman. A coding theorem for distributed computation. In *STOC '94: Proceedings of the Twenty-sixth Annual ACM Symposium on Theory of Computing*, pages 790–799. ACM, 1994.

[74] Leonard J. Schulman. Communication on noisy channels: A coding theorem for computation. *Foundations of Computer Science, Annual IEEE Symposium on*, pages 724–733, 1992.

[75] Leonard J. Schulman. Deterministic coding for interactive communication. In *STOC '93: Proceedings of the Twenty-fifth Annual ACM Symposium on Theory of Computing*, pages 747–756, 1993.

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

[77] Leonard J. Schulman. A postcript to "coding for interactive communication". [Online:] `http://www.cs.caltech.edu/~schulman/Papers/intercodingpostscript.txt`, 2003.

[78] Adi Shamir. IP = PSPACE. *J. ACM*, 39(4):869–877, October 1992.

[79] Claude E. Shannon. The zero error capacity of a noisy channel. *Information Theory, IRE Transactions on*, 2(3):8–19, September 1956.

[80] Claude E. Shannon. A mathematical theory of communication. *ACM SIGMOBILE Mobile Computing and Communications Review*, 5(1):3–55, 2001. Originally appeared in *Bell System Tech. J.* 27:379–423, 623–656, 1948.

[81] Alexander A. Sherstov and Pei Wu. Optimal interactive coding for insertions, deletions, and substitutions. *Electronic Colloquium on Computational Complexity (ECCC)*, 24:79, 2017.

[82] John M. Wozencraft. List decoding. Technical report, Research Laboratory of Electronics, Massachusetts Institute of Technology. Quarterly Progress Report, no. 48, 1958.

[83] Andrew C.-C. Yao. Some complexity questions related to distributive computing(preliminary report). In *Proceedings of the Eleventh Annual ACM Symposium on Theory of Computing*, STOC '79, pages 209–213, 1979.

[84] Andrew C.-C. Yao. Quantum circuit complexity. In *Foundations of Computer Science, 1993. Proceedings., 34th Annual Symposium on*, pages 352–361, 1993.

Index

adaptive coding schemes, 38–44
adversarial noise, 4

binary entropy, 45
broadcast channel, 94
binary symmetric channel (BSC), 4

capacity, 52, 64, 69
coding scheme, 2
completeness, 88
confirmation bits, 60
consistency check, 17, 19, 46

erasure channel, 4, 67–70
erasures, 21, 44, 67

feedback, 4, 55–67

Hamming distance, 8
hash
    hash collision, 19, 51
    hash functions, 17, 47

insertions and deletions, 4, 70
interactive list-decoding, 32
    verifiable list-decoding schemes, 33
interactive proofs, 88
interactive protocol, 2

meeting points, 19, 49
multiparty coding scheme, 5, 74–82

noise rate, 3
    relative noise rate, 41, 43
noise-resilient circuits, 84
nonadaptive protocols, 20, 58

optimal resilience, 12–16, 20
    binary coding scheme, 16
    feedback channel, 56–64
order of speaking, 5, 20, 43–44, 52

potential function, 18, 30, 51
privacy, 86
protocol tree, 12

rate, 3, 45
    constant rate, 3, 76–78
    high rate coding schemes, 45–54, 64–67, 69
    vanishing rate, 3, 23–26, 28, 41–44
repetition code, 74
rewind-if-error, 16–19, 45–52, 56–70, 81

shared randomness, 5, 21, 47, 49, 90
silence encoding, 43
soundness, 88
suffix distance, 9, 32, 71

tree codes, 7–12
    edit distance tree codes, 70
    efficiency, 23
    encoding and decoding, 11
    linear tree codes, 11, 92

list tree codes, 32
local tree codes, 26
potent tree codes, 28
systematic tree codes, 92
tree code based schemes, 23–37, 70, 74, 89

zero knowledge, 86