Electronic Colloquium on Computational Complexity, Revision 1 of Report No. 43 (2012)

Efficient Interactive Coding Against Adversarial Noise

Zvika Brakerski[*] Yael Tauman Kalai[†]

Abstract

In this work, we study the problem of constructing interactive protocols that are robust to noise, a problem that was originally considered in the seminal works of Schulman (FOCS '92, STOC '93), and has recently regained popularity. Robust interactive communication is the interactive analogue of error correcting codes: Given an interactive protocol which is designed to run on an error-free channel, construct a protocol that evaluates the same function (or, more generally, simulates the execution of the original protocol) over a noisy channel. As in (non-interactive) error correcting codes, the noise can be either stochastic, i.e. drawn from some distribution, or adversarial, i.e. arbitrary subject only to a global bound on the number of errors.

We show how to *efficiently* simulate any interactive protocol in the presence of constant-rate *adversarial* noise, while incurring only a constant blow-up in the communication complexity ($\mathtt{CC}$). Our simulator is randomized, and succeeds in simulating the original protocol with probability at least $1 - 2^{-\Omega(\mathtt{CC})}$. (Prior works could not achieve efficient simulation in the adversarial case.)

[*] Stanford University, `zvika@stanford.edu`. Supported by a Simons Postdoctoral Fellowship and by DARPA.

[†] Microsoft Research, `yael@microsoft.com`.

ISSN 1433-8092

1 Introduction

The problem of errors in communication is very fundamental, and modern study of this problem dates back to the work of Shannon [Sha48]. Two types of errors are typically considered: Stochastic errors, which are assumed to be distributed according to some probability distribution; and adversarial errors, which are not governed by a probability distribution, but rather by the whims of an adversary. Today we know how to construct "good" error correcting codes: ones that encode k-bit messages using $O(k)$-bit codewords, and are decodable even with $\Omega(k)$ adversarial errors. In other words, codes with constant *information rate* and constant *error rate*. For more information on error correcting codes, see e.g. [Sud01] and references within.

In this work, we study the *interactive* analogue of error-correcting codes, a problem that was originally considered in an innovative sequence of works by Schulman [Sch92, Sch93]. We show how to efficiently convert any interactive protocol into one which is resilient to constant fraction of adversarial error, while increasing the communication complexity by at most a constant, and by increasing the computational complexity by at most a polynomial factor. (See Section 1.2 for more details on our results.)

It may seem that interactive error correction is implied by traditional (non-interactive) error correcting codes, by simply encoding each message of the interactive protocol using the code. However, this solution is often unsatisfactory. In the case of adversarial noise, the resulting error-rate will be very small. In particular, the error-rate will be smaller than $1/k$, where k is the number of messages transmitted in the protocol. This is because we cannot allow even a single message to be fully corrupted. In the stochastic case, we can use sufficiently long codes to achieve reasonable error rate. However, if the parties exchange many relatively short messages (say, one bit each), then the code will incur an undesirable large (super-constant) blowup in the communication complexity.

1.1 Related Work

As mentioned above, the study of error-resilient interactive communication started with a sequence of results by Schulman, where his objective was to construct a simulator (or a compiler), that given any interactive protocol π constructs an error-resilient version of π, that is robust to constant fraction of errors, and which has only a constant overhead in the communication complexity $\mathtt{CC}$.

Schulman's works offer solutions in a number of scenarios: First, in [Sch92] it is shown that if the parties communicate over a probabilistic channel (specifically, a BSC: binary symmetric channel) and share a large amount ($O(\mathtt{CC}^2)$) of random bits, then the objective can indeed be met. Furthermore, in this case the resulting robust protocol is as efficient as the original one (up to polynomial factors). The pre-shared randomness could be replaced with local randomness at the cost of making the simulator inefficient. Then, in [Sch93], tree-codes were introduced: If the communicating parties are given access to a tree code of depth $O(\mathtt{CC})$, then even the more challenging task of adversarial errors can be tackled deterministically. Schulman showed that such tree codes exist, but was unable to show an efficient implementation (even randomized).

Braverman and Rao [BR11] significantly improved the error rate in Schulman's work, and showed that tree codes can be used to protect against error rate of up to 1/8 (in the binary case). They also provided some evidence that protecting against error rate 1/4 should be challenging. Their construction again inherits the inefficiency of the tree constructions. Consequently, Braverman [Bra12] showed how to construct and decode tree-codes in sub-exponential time, thus reducing the computation complexity of previous works from exponential time to sub-exponential time.

Gelles, Moitra and Sahai [GMS11] gave an efficient way to convert any interactive protocol into an error resilient one in the case of BSC (probabilistic channels). They use a relaxed notion of tree codes, which they show can be efficiently constructed and decoded over BSC.

Despite all this remarkable progress, the question of an efficient solution for adversarial channels remained open, and is addressed in this work.

1.2 Our Results

We present an efficient randomized simulator for interactive protocols over channels with adversarial noise. Naturally, our solution can also be applied towards probabilistic channels where the total number of errors is bounded with high probability. The properties of our simulator are described in the following theorem. (All of our protocols are over the binary alphabet.)

Theorem. *There exists an efficient simulator S such that for any protocol $\pi = (A, B)$ of communication complexity $\mathtt{CC}(\pi)$, the simulated protocol $S^\pi = (S^A, S^B)$ computes $\mathsf{Trans}(\pi)$ (the transcript of π), has communication complexity $O(\mathtt{CC}(\pi))$, and is robust, with probability $\left(1 - 2^{-\Omega(\mathtt{CC}(\pi))}\right)$, to adversarial errors of rate $\frac{(1-\epsilon)\cdot\eta}{4}$, where η is the highest inefficiently-correctable error rate. The computational complexity of the simulator (given oracle access to the parties) is at most* $\mathrm{poly}(\mathtt{CC}(\pi))$.

As in previous works, the convergence to the asymptotic error rate is inversely proportional to the overall communication complexity. Namely, to achieve error rate $(1-\epsilon)\eta/4$, the communication complexity of the simulator will be $O\left(\frac{\mathtt{CC}(\pi)}{\epsilon}\right)$.

The best currently known asymptotic value of η is 1/8 due to [BR11], which implies asymptotic rate of 1/32 for our simulator. The factor 1/4 loss in the error rate is a technical artifact of our analysis. While this is the best that we could extract from our methods, we do not see any barrier towards improving this factor.

Our simulator is inherently randomized due to our use of hash functions (see Section 1.3 below for details). This implies that even when the input protocol π is deterministic, S^π will be probabilistic, which might be a drawback in some situations. In contrast, the inefficient simulators of [Sch93, BR11] are deterministic. The question of finding an efficient deterministic simulator remains a very interesting open problem.

1.3 Our Techniques

Our starting point is the aforementioned exponential time deterministic simulator from either [Sch96, BR11, Bra12], that we use in a black-box manner.[1] In order to use such a simulator and still achieve overall computational efficiency, we will use the simulator on logarithmic chunks of the protocol.

The basic idea is simple: Given a protocol $\pi = (A, B)$ with an N bit transcript, divide its transcript into $N/\log N$ chunks of $\log N$ bits each. Then run the exponential simulator chunk-by-chunk to reconstruct the entire transcript of π efficiently. This idea indeed seems to work in the stochastic model, where the errors are distributed roughly evenly between the different chunks (though the failure probability will not be negligible).

[1] This means that any future simulator, even non-tree based, can be used. In fact, we can also use probabilistic simulators with exponentially small error probability, but it would complicate the analysis somewhat.

However, in the adversarial setting this idea is prone to failure, since we are only guaranteed that the *average* error rate over the chunks is constant, but it can be very high for any particular chunk. Namely, if the adversary introduces high noise rate at a specific chunk, then it can make the parties get that chunk all wrong, which will ruin correctness, even if all other chunks are computed correctly. We thus have to introduce some control mechanism by which the parties can identify that such erroneous event occurred, rewind their state back to the point of agreement, and try again. (This is the high level logic governing all known solutions starting from [Sch92].)

As a first solution, we introduce a synchronization check before each chunk, where the two parties compare their internal states to see that they are in sync. String comparison is known to be efficiently possible using (private, non-shared) randomness: each party will draw a randomness-efficient universal hash function, apply it to its local copy of the (simulated) transcript, and send the outcome, together with the description of the hash function, to the other party.[2] Using randomness efficient hash families [NN93, AGHP92], only $O(\log N)$ bits need to be sent to achieve good detection probability, and since the check only happens once for every $\log N$ bits chunk, its amortized effect on the information rate is constant.

This solution, however, does not work as is: The adversary might realize that the string comparison is the soft underbelly of our construction, and introduce errors during that stage. On the face of it, even a small amount of error can completely ruin the comparison.

We overcome this problem by incorporating the synchronization check as part of the chunk that is being communicated. Namely, we consider a protocol that first sends (and receives) the synchronization information, and then the parties run the next chunk. We apply the exponential time simulator to this extended protocol, whose communication complexity is still logarithmic. This will ensure that an adversary who wants to cause harm at any part of the protocol needs to introduce at least $\Omega(\log N)$ errors (a constant fraction of the communication).

The parties will simulate this extended protocol (synchronization + next chunk) over the channel and end up with a transcript, which contains the information of whether they are in sync or not, as well as the $\log N$ bits of transcript corresponding to the current chunk. If they were in sync, then they will use the $\log N$ bits of transcript, and continue to the next chunk. If they were not in sync, then they discard this information, and go back to the previous chunk (to try to get in sync).

Still there is a problem, since the adversary can adopt the following line of attack: It can invest enough errors to corrupt the view of *only one party*, and corrupt the check accordingly. In such case, one party will revert to the previous chunk, while the other continues to the next. The protocol we described so far gives no mechanism to help the parties verify that they are computing the same chunk.

We therefore add an additional element to the synchronization check: in addition to the hash description and hash value, each party will also send its position in the simulated transcript, which comes at a tolerable cost of additional $O(\log N)$ bits (of course this is also incorporated into the protocol that is "protected" by the exponential simulator). Given this information, it is possible to efficiently detect and correct gaps.

To analyze this protocol, we can think about the error correcting protocol as a game where we try to make the adversary waste its allotted number of errors, without setting the protocol back by too much. Intuitively, so long as the adversary only sets us back by less than the amount of steps required to recuperate (up to a constant), then our simulator will succeed. In our protocol, the

[2] Interestingly, [Sch92] also uses universal hashing to compare the internal state, however he could not afford to send the description of the hash function along with the output, so he had to use pre-shared randomness.

adversary needs to invest $\Omega(\log N)$ errors to create an initial inconsistency, and it needs to keep investing $\Omega(\log N)$ errors in each following step to prevent our synchronization mechanism from recovering. Our analysis shows that the adversary will run out of errors at some point, allowing our recovery mechanisms to complete the simulation of the transcript of π.

The last risk that remains is that the adversary might corrupt the final chunks, leaving no time for recovery. This is treated similarly to previous works: We pretend that the transcript is actually longer than it really is by padding it with zeros. This way, a corruption at the end of the simulation can only harm the padding, which is thrown away anyway.

Our simulator is formally presented and analyzed in Section 3.

2 Preliminaries

The following simple claim is used in our analysis.

Claim 2.1. *Let $X_1, \ldots, X_n$ be i.i.d. random variables over $\{0,1\}$ with expectation $\mathbb{E}[X_i] = 1 - \epsilon$, and let $X = \sum_{i \in [n]} X_i$. Then for any integer t,*

$$\Pr[X \le n - t] \le 2^n \cdot \epsilon^t \ .$$

Proof. For $t < 0$ or $t > n$, the claim holds trivially. Otherwise, the event $X \le n - t$ is exactly the event that at least t variables take a 0 value, therefore

$$\Pr[X \le n - t] = \sum_{i=t}^{n} \binom{n}{i} \epsilon^i (1-\epsilon)^{n-i} \le \sum_{i=t}^{n} \binom{n}{i} \epsilon^t \le 2^n \cdot \epsilon^t \ ,$$

and the claim follows. □

An immediate corollary is that Claim 2.1 also holds when t is any real number (by applying the original claim to $\lceil t \rceil$). This corollary is used in the proof of Lemma 3.4.

Let X, Y be real random variables. We say that $Y \ge X$ if $F_Y(z) \le F_X(z)$ for all z, where $F_X(z) = \Pr[X \le z]$ is the cumulative distribution function of X.

2.1 Interactive Protocols and Simulation

An interactive protocol is defined by a pair of interactive machines $\pi = (A, B)$. We denote the transcript of a protocol by $\mathsf{Trans}(\pi)$ and the communication complexity of the protocol (= length of the transcript) by $\mathtt{CC}(\pi)$. In this work, we consider simulators for interactive communication. A simulator is an oracle machine S, such that (S^A, S^B) results in both parties obtaining the transcript of the original protocol $\mathsf{Trans}(\pi)$, in spite of channel errors. Clearly, it is sufficient to simulate deterministic protocols with no input, since we can always hard-wire the randomness and input into the protocol. Note that in such case $\mathsf{Trans}(\pi)$ is unique.

All of the protocols we consider in this work are over the binary alphabet. For the sake of concreteness, we assume a model where at each round of the protocol, both A, B simultaneously send one bit over the channel (this model was used in previous works). However, our results extend without any changes to the case where each party speaks in turn.

Given a transcript T, we let $X(T)$ denote the interactive machine whose initial internal state is the internal state of X right after producing T (while interacting with another interactive machine).

If X cannot produce T or if X halts, then we define $X(T) \equiv 0$. Note that it is easy to simulate $X(T)$ given X and T, by feeding T to X one message at a time until X is at the right internal state.

Given a (partial) transcript T and integer n, we let $T_{\leq n}$ denote the n bit prefix of T. Concatenation of two transcripts is denoted by $T_1 \| T_2$.

The following theorem asserts the existence of deterministic exponential time simulators for any interactive protocol.

Theorem 2.2 ([Sch96, BR11, Bra12]). *There exist positive constants $\rho, \eta \in (0,1)$ and a deterministic interactive oracle machine Q (the simulator) such that for any protocol $\pi = (A, B)$ of communication complexity $\mathtt{CC}(\pi)$, the protocol $Q^\pi = (Q^A, Q^B)$ computes $\mathsf{Trans}(\pi)$, has communication complexity $\mathtt{CC}(Q^\pi) \leq \mathtt{CC}(\pi)/\rho$, and is robust (with probability 1) to adversarial error of rate η. The computational complexity of Q is at most $2^{O(\mathtt{CC}(\pi))}$.*[3]

For the sake of simplicity, and w.l.o.g., we assume that Q always terminates after communicating the same number of bits, and always outputs a bit string of length $\mathtt{CC}(\pi)$, even when it is unsuccessful due to noise rate higher than η or due to the oracle aborting.

2.2 Hash Functions

Our simulator uses a randomness-efficient string comparison test. Such is provided by using the families of hash functions of either [NN93, AGHP92] (see in particular [NN93, Section 9]). In what follows, we denote $\{0,1\}^{\leq n} \triangleq \cup_{i \in [n]} \{0,1\}^i$.

Theorem 2.3 ([NN93, AGHP92]). *There exists a constant $q > 0$ and an ensemble of hash families $\{\mathcal{H}_k\}_{k \in \mathbb{N}}$ such that for every $k \in \mathbb{N}$ and for every $h \in \mathcal{H}_k$, $h : \{0,1\}^{\leq 2^k} \to \{0,1\}^{q \cdot k}$ is poly-time computable, it is efficient to sample $h \leftarrow \mathcal{H}_k$ using only $q \cdot k$ random bits, and for all $x \neq y \in \{0,1\}^{\leq 2^k}$ it holds that*

$$\Pr_{h \leftarrow \mathcal{H}_k}[h(x) = h(y)] \leq 2^{-k} \ .$$

Note that we assume w.l.o.g. that the seed length and output size of the hash function are identical (otherwise, define q according to the maximal of the two).

3 Error-Resilient Interactive Protocols

In this section, we show how to convert any interactive protocol $\pi = (A, B)$ into one that is resilient to constant fraction of adversarial errors, and is *efficient* in the sense that it computes $\mathsf{Trans}(\pi)$ with a polynomial overhead in the time complexity. Recall that without loss of generality A, B are deterministic and don't take any input.

More specifically we present an efficient simulator S that has oracle access to either A or B, and simulates these parties in an error resilient manner, so that the new protocol (S^A, S^B) computes

[3] This computational complexity also takes into account the construction of a tree code of depth $O(\mathtt{CC}(\pi))$. Specifically, Schulman [Sch96] proved the existence of the required tree codes for any depth d using the probabilistic method, and using $O(d)$ bits of randomness. Since tree code properties can be verified in time polynomial in their size (exponential in the depth), a deterministic exponential time tree construction algorithm follows by going over all possible random strings.

$\mathsf{Trans}(\pi)$, even in the presence of constant fraction of adversarial errors. We often use X to indicate one of $\{A, B\}$, in which case Y will denote the other party.

The simulator S is presented in Section 3.1 and analyzed in Section 3.2.

3.1 Simulator S^{π}

Let N be an upper bound on the communication complexity of $\pi = (A, B)$, such that N is a power of 2. Throughout the simulation, party X maintains a local variable T_X that represents its current view on the partial transcript of π. At the end of the algorithm, the N-prefix of T_X will be equal to $\mathsf{Trans}(\pi)$.

We consider the simulator Q from Theorem 2.2 with parameters ρ, η, and the hash family from Theorem 2.3 with parameter q. We define a parameter τ which will be useful in the presentation of the algorithm and in the analysis:

$$\tau \triangleq \frac{4q+3}{\rho} \ . \tag{1}$$

As outlined in Section 1.3, our simulator $S^{\pi} = (S^A, S^B)$ (Figure 1) works in $O(N/\log N)$ rounds. Each round contains an execution of a logarithmic-communication subroutine Chunk (Figure 2), which is "protected" from channel errors using the exponential time simulator Q. (Of course this protection can sometimes fail when there are too many errors, but we will show that the process converges nonetheless.) In total, each round communicates $\tau \log N$ bits. We elaborate more on the subroutine and the use of Q, below.

The subroutine Chunk communicates the synchronization information, as well as (what it believes to be) the next chunk of the transcript between the two parties. This subroutine is never "really" executed, but rather simulated by the exponential time simulator Q (Chunk only communicates $O(\log N)$ bits). The simulator Q returns a "protected transcript" of the execution of Chunk. This transcript may be that of a legal execution if there were not too many errors, or it can be completely arbitrary if there were.

Given the protected transcript, the parties can check if their states are in sync. If not, they can move towards rectifying the situation. If they were in sync, then the protected transcript indeed contains the next chunk of $\mathsf{Trans}(\pi)$, which can be appended to the local copy of the transcript. Of course there is always the chance that the protected transcript is wrong, but our analysis shows that this does not set us back by too much.

Lastly, a parameter c controls the number of rounds of our simulator. The value of c determines the convergence of S to its asymptotic tolerable error rate. The larger c is, the closer the simulator gets to tolerating $\eta/4$ fraction of errors. We will assume w.l.o.g. that $cN/\log N$ is integer.

Since the typical value for c is a large constant (in fact, Theorem 3.1 is meaningless unless $c > 5$), the local transcripts that the parties maintain are longer than N. The output of the simulator will be the N-bit prefix of this long local transcript. Therefore, if the adversary corrupts the last round of the execution, it will not affect the output, which is the N-bit prefix of the local transcripts.

The simulator is presented in Figure 1. The subroutine Chunk is presented in Figure 2.

3.2 Analysis

The following theorem summarizes the properties of our simulator.

```
Simulator $S^X$

• Input: Oracle access to interactive machine $X$.
• Output: Transcript $T \in \{0,1\}^N$.
• Operation:
    1. Set $T{:=}\phi$.
    2. Repeat $cN/\log N$ times:
        (a) Sample a new hash function $h_x \leftarrow \mathcal{H}_{\log(cN)}$ (recall Theorem 2.3), and set $\sigma_x{:=}h_x(T)$.
        (b) Let $X'$ be shorthand notation for the subroutine $\mathsf{Chunk}$ with the current variable values $(T, h_x, \sigma_x)$. Formally: $X' \triangleq \mathsf{Chunk}^X_{T,h_x,\sigma_x}$.
        (c) Use the simulator $Q^{X'}$ to simulate $X'$. The output of $Q$ is a simulated transcript of the form: $(i, \tilde{h}_x, \tilde{\sigma}_x) \| (j, h_y, \sigma_y) \| L$, where $|L| = \log N$.
        (d) If $(i \neq |T|/\log N)$ or $(\tilde{h}_x \neq h_x)$ or $(\tilde{\sigma}_x \neq \sigma_x)$, then finish this iteration.
            Otherwise we proceed with one of the following cases:
            • If $(i > j)$ then set $T{:=}T_{\leq (i-1)\log N}$.
            • If $(i < j)$ then finish this iteration.
            • If $((i = j)$ and $(h_y(T) \neq \sigma_y))$ then set $T{:=}T_{\leq (i-1)\log N}$.
            • If $((i = j)$ and $(h_y(T) = \sigma_y))$ then set $T{:=}T\|L$.
    3. Output $T_{\leq N}$.
```

Figure 1: Our simulator.

```
Subroutine $\mathsf{Chunk}^X_{T,h_x,\sigma_x}$

  i. Let $i{:=}|T|/\log N$, represented as a bit string of length $\log N$.
 ii. Send $(i, h_x, \sigma_x)$ over the channel and receive $(j, h_y, \sigma_y)$. (This is of course done bit by bit.)
iii. Execute $X(T)$ for $\log N$ communication steps.
```

Figure 2: Subroutine to be fed into Q.

Theorem 3.1. *For any protocol $\pi = (A, B)$ of communication complexity $\mathtt{CC}(\pi)$, the protocol $S^\pi = (S^A, S^B)$ computes $\mathsf{Trans}(\pi)$, has communication complexity $\mathtt{CC}(S^\pi) = O(\mathtt{CC}(\pi))$, and is robust with probability $\left(1 - 2^{-\Omega(\mathtt{CC}(\pi))}\right)$ to adversarial channels of error rate $(1 - 5/c) \cdot (\eta/4)$. The computational complexity of S is at most* $\mathrm{poly}(\mathtt{CC}(\pi))$.

The theorem follows by combining Lemmas 3.2, 3.3, and Corollary 3.5 of Lemma 3.4, below. In what follows, recall that the constant q is from Theorem 2.3, the constants ρ and η are from Theorem 2.2, and the constant τ is defined in Eq. (1).

Lemma 3.2. *It holds that*

$$\mathtt{CC}(S^\pi) \leq c\tau N = O(\mathtt{CC}(\pi)) \ .$$

Proof. The protocol S^π is composed of $c \cdot N/\log N$ rounds, each containing a simulation of X' which communicates $\frac{2(2q+1)\log N + \log N}{\rho} = \tau \log N$ bits. The communication complexity is therefore:

$$\mathtt{CC}(S^\pi) = (cN/\log N) \cdot (\tau \log N) = c\tau N = O(N) \ . \qquad \square$$

Lemma 3.3. *The computational complexity of S^π is at most* $\mathrm{poly}(\mathtt{CC}(\pi))$.

Proof. The simulator S runs in $c \cdot N/\log N = O(N/\log N)$ rounds. In each round, the simulator Q is called on a machine X' that communicates $2(2q+1)\log N + \log N = (4q+3)\log N$ bits. The computational complexity of Q on such machines is $\mathrm{poly}(N)$. In addition, each party makes two evaluations of hash functions, which contribute additional $\mathrm{poly}(N)$ computational steps. All of the other operations are simple manipulations on the transcript. □

We can make the analysis above more specific: Letting t_h denote the time complexity of the function family $\mathcal{H}_{\log(cN)}$; and letting t_Q denote the time complexity of Q when executed on protocols of communication complexity $(4q+3)\log N$, we get that the computational complexity of S^π is at most $O((N/\log N) \cdot (t_h + t_Q + \log N))$ (in some reasonable computational model). Using known instantiations, we can get $t_h = \Theta(N)$ (note that the hash function must read all of its input). We are not aware of a precise analysis as to the running time of Q, so we can only say that $t_Q = \mathrm{poly}(N)$ for an unspecified polynomial.

The following lemma proves the success probability of our simulation over adversarial noisy channels. This is the heart of our analysis which is used to derive Corollary 3.5 below.

Lemma 3.4. *When running S^π over a channel that makes at most*

$$E = (c-5) \cdot \frac{\eta\tau}{4} \cdot N$$

adversarial errors, S^π outputs $\mathsf{Trans}(\pi)$ with probability at least $1 - 2^{-(1-o(1))N}$.

Proof. Let $\pi = (A, B)$ and consider an execution of $S^\pi = (S^A, S^B)$. For $X \in \{A, B\}$, we denote the values computed by S^X during the protocol with subscript X (e.g., T_A or i_B). We denote $i_X \triangleq |T_X| / \log N$ (note that this is always an integer).

We define the following (random) variables:

- Good transcript prefix (in chunks) g: This is the longest common prefix of T_A, T_B, rounded to whole chunks. Namely, if g' is the longest common prefix in bits, then $g = \lfloor g'/\log N \rfloor$.

- Gap values α_A, α_B: We define $\alpha_X \triangleq i_X - g$ (naturally, α_X is always non-negative).

- Error count e: This is the number of errors the adversary injected into the channel so far.
- Potential: We define a potential function

  $$\varphi \triangleq (g - \alpha_A - \alpha_B) \cdot \log N + \frac{4}{\eta\tau} \cdot e \ ,$$

  where τ is as defined in Eq. (1).

We show that when the algorithm terminates, it holds, with probability $1 - 2^{-(1-o(1))N}$, that

$$\varphi \geq (c-4)N \ .$$

This implies that

$$g \log N \geq \varphi - \frac{4}{\eta\tau} \cdot E \geq (c-4)N - (c-5)N = N \ ,$$

which in turn means that the two parties agree on the prefixes $T_{A,\leq N} = T_{B,\leq N} = \mathsf{Trans}(\pi)$.

We remark that the coefficient 4 in the definition of the potential function is "responsible" for the loss in the error rate of our simulator ($\eta/4$ compared to η). In the course of presenting our analysis, we will explain what warrants this factor.

Our proof follows by showing that the potential function φ must grow by roughly $\log N$ with every round of the protocol. Let φ_ℓ denote the change in φ in iteration ℓ (where $\ell = 1, \ldots, cN/\log N$). We use case analysis to lower-bound φ_ℓ:

- **Case 1:** The number of errors in the iteration is at most $\eta\tau \log N$. In this case we are guaranteed that Q simulates A', B' correctly (where A' is the machine X' defined by S^A in Step 2b, and B' is the same for S^B). Therefore the output of Q (for both parties) is the real transcript of the protocol (A', B').

  Again we have a few cases.

  - If $i_A \neq i_B$, then both parties will get $i \neq j$. In such case, it must be that for some X, $\alpha_X > \alpha_Y \geq 0$. The larger α_X necessarily belongs to the party with the larger i_X, and this party will chop a chunk off its transcript. We conclude that $\varphi_\ell \geq \log N$.
  - If $i_A = i_B$ and $T_A = T_B$, then this means that both partial transcripts agree. In this case, L is indeed the next chunk of the execution and both parties will append it to their transcripts. We conclude that $\varphi_\ell \geq \log N$.
  - If $i_A = i_B$ and $T_A \neq T_B$, then the outcome depends on the randomness of the hash functions. Let D_ℓ denote the event that $h_A(T_A) \neq h_B(T_B)$. Then by Theorem 2.3 and the union bound, D_ℓ happens with probability at least $1 - 2/(cN)$.

    If D_ℓ happens, then both α_A, α_B decrease by 1 and $\varphi_\ell \geq 2 \log N$. If D_ℓ doesn't happen, then one or two of the parties might append a faulty L, causing α_A, α_B to increase by 1, namely $\varphi_\ell \geq -2 \log N$. We conclude that $\varphi_\ell \geq 2 \log N(-1 + 2 \cdot \mathbb{1}_{D_\ell})$.

- **Case 2:** The number of errors in the iteration is greater than $\eta\tau \log N$. In this case, all bets are off and the expression $(g - \alpha_A - \alpha_B)$ can decrease by at most 3 (the worst case is when g decreases by 1 and i_X increases by 1, causing α_X to increase by 2, note that $\alpha_Y = 0$ in this case). We conclude that

  $$\varphi_\ell \geq -3 \log N + \frac{4}{\eta\tau} \cdot (\eta\tau) \log N \geq \log N \ .$$

The above equation explains the need for a factor 4 in the definition of the potential function, which is responsible for the loss in error rate. (One could think that any factor larger than 3 should be sufficient, but we found it problematic for other parts of the analysis.)

We conclude that either $\varphi_\ell \geq \log N$ or $\varphi_\ell \geq 2\log N(-1+2\cdot \mathbb{1}_{D_\ell})$. Let k be the number of rounds for which the latter holds. Then

$$\begin{aligned}\varphi &= \sum_{\ell=1}^{cN/\log N} \varphi_\ell \\ &\geq (cN/\log N - k)\log N + \sum_{\ell'=1}^{k} 2\log N(-1+2\cdot \mathbb{1}_{D_{\ell'}}) \\ &= cN - 3k\log N + 4\log N \sum_{\ell'=1}^{k} \mathbb{1}_{D_{\ell'}} \ .\end{aligned}$$

Since the variables $D_{\ell'}$ are independent, then recalling Claim 2.1, we get

$$\Pr\left[\sum_{\ell'=1}^{k} \mathbb{1}_{D_{\ell'}} \leq k - N/\log N\right] \leq 2^k \cdot (2/(cN))^{N/\log N} \leq 2^{(c+1-\log c)N/\log N} \cdot 2^{-N} = 2^{-(1-o(1))N} \ .$$

If the above bad event does not happen, then

$$\varphi \geq cN - 3k\log N + 4k\log N - 4N \geq (c-4)N \ ,$$

and the lemma follows. □

Finally, the error rate for which robustness holds follows immediately.

Corollary 3.5. *S^π is robust with probability $\left(1-2^{-(1-o(1))N}\right)$ to adversarial channels of rate*

$$\left(1-5/c\right)\cdot\frac{\eta}{4} \ .$$

Proof. Combining Lemma 3.2 and Lemma 3.4, it follows that with probability $1-2^{-(1-o(1))N}$, the protocol S^π is robust to noise rate

$$\frac{E}{\mathtt{CC}(S^\pi)} = \frac{(c-5)\eta\tau N}{4\cdot c\tau N} = (1-5/c)\cdot\frac{\eta}{4} \ .$$ □

Acknowledgments

We thank Moni Naor for his insightful comments on an earlier version of this manuscript. We also thank anonymous FOCS 12 reviewers for their comments.

References

[AGHP90] Noga Alon, Oded Goldreich, Johan Håstad, and René Peralta. Simple constructions of almost k-wise independent random variables. In *FOCS*, pages 544–553. IEEE Computer Society, 1990.

[AGHP92] Noga Alon, Oded Goldreich, Johan Håstad, and René Peralta. Simple construction of almost k-wise independent random variables. *Random Struct. Algorithms*, 3(3):289–304, 1992. Journal version of [AGHP90].

[BR11] Mark Braverman and Anup Rao. Towards coding for maximum errors in interactive communication. In Fortnow and Vadhan [FV11], pages 159–166.

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

[FV11] Lance Fortnow and Salil P. Vadhan, editors. *Proceedings of the 43rd ACM Symposium on Theory of Computing, STOC 2011, San Jose, CA, USA, 6-8 June 2011*. ACM, 2011.

[GMS11] Ran Gelles, Ankur Moitra, and Amit Sahai. Efficient and explicit coding for interactive communication. In Rafail Ostrovsky, editor, *FOCS*, pages 768–777. IEEE, 2011. Preliminary versions in [GS11, Moi11].

[GS11] Ran Gelles and Amit Sahai. Potent tree codes and their applications: Coding for interactive communication, revisited. *CoRR*, abs/1104.0739, 2011.

[Moi11] Ankur Moitra. Efficiently coding for interactive communication. *Electronic Colloquium on Computational Complexity (ECCC)*, 18:42, 2011.

[NN90] Joseph Naor and Moni Naor. Small-bias probability spaces: Efficient constructions and applications. In Harriet Ortiz, editor, *STOC*, pages 213–223. ACM, 1990.

[NN93] Joseph Naor and Moni Naor. Small-bias probability spaces: Efficient constructions and applications. *SIAM J. Comput.*, 22(4):838–856, 1993. Journal version of [NN90].

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

[Sch93] Leonard J. Schulman. Deterministic coding for interactive communication. In S. Rao Kosaraju, David S. Johnson, and Alok Aggarwal, editors, *STOC*, pages 747–756. ACM, 1993.

[Sch96] Leonard J. Schulman. Coding for interactive communication. *IEEE Transactions on Information Theory*, 42(6):1745–1756, 1996. Journal version of [Sch92, Sch93] (refers mostly to the latter).

[Sha48] C. E. Shannon. A mathematical theory of communication. *The Bell Systems Technical Journal*, 27:379–423, 623–656, 1948.

[Sud01] Madhu Sudan. Algorithmic introduction to coding theory – class lecture notes, fall 2001. `http://people.csail.mit.edu/madhu/FT01/course.html`.

ECCC ISSN 1433-8092

http://eccc.hpi-web.de