Efficient Multiparty Interactive Coding for Insertions, Deletions, and Substitutions

[Extended Abstract]*

Ran Gelles[†]
Bar-Ilan University
ran.gelles@biu.ac.il

Yael Tauman Kalai
Microsoft Research & MIT
yael@microsoft.com

Govind Ramnarayan[‡]
MIT
govind@mit.edu

ABSTRACT

In the field of interactive coding, two or more parties wish to carry out a distributed computation over a communication network that may be noisy. The ultimate goal is to develop efficient coding schemes that can tolerate a high level of noise while increasing the communication by only a constant factor (i.e., constant rate).

In this work we consider synchronous communication networks over an arbitrary topology, in the powerful adversarial insertion-deletion noise model. Namely, the noisy channel may adversarially alter the content of any transmitted symbol, as well as completely remove a transmitted symbol or inject a new symbol into the channel.

We provide efficient, constant rate schemes that successfully conduct any computation with high probability as long as the adversary corrupts at most $\frac{\epsilon}{m}$ fraction of the total communication, where m is the number of links in the network and ϵ is a small constant. This scheme assumes an *oblivious* adversary which is independent of the parties' inputs and randomness. We can remove this assumption and resist a worst-case adversary at the price of being resilient to $\frac{\epsilon}{m \log m}$ errors.

While previous work considered the insertion-deletion noise model in the two-party setting, to the best of our knowledge, our scheme is the first multiparty scheme that is resilient to insertions and deletions. Furthermore, our scheme is the first *computationally efficient* scheme in the multiparty setting that is resilient to *adversarial* noise.

CCS CONCEPTS

• **Theory of computation** → **Interactive computation; Error-correcting codes; Distributed algorithms.**

KEYWORDS

interactive coding, distributed algorithms, error correcting codes

*See [16] for a longer version of this work.

[†]Supported in part by the Israel Science Foundation (ISF) through grant No. 1078/17.

[‡]Supported in part by NSF award CCF 1665252 and NSF award DMS-1737944.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

PODC '19, July 29–August 2, 2019, Toronto, ON, Canada

© 2019 Association for Computing Machinery.

ACM ISBN 978-1-4503-6217-7/19/07...\$15.00

<https://doi.org/10.1145/3293611.3331621>

ACM Reference Format:

Ran Gelles, Yael Tauman Kalai, and Govind Ramnarayan. 2019. Efficient Multiparty Interactive Coding for Insertions, Deletions, and Substitutions: [Extended Abstract]. In *2019 ACM Symposium on Principles of Distributed Computing (PODC'19), July 29–August 2, 2019, Toronto, ON, Canada*. ACM, New York, NY, USA, 10 pages. <https://doi.org/10.1145/3293611.3331621>

1 INTRODUCTION

In his pioneering work, Schulman [28, 29] introduced and studied the problem of how to perform an interactive two-party computation over a noisy communication channel. In [27], Rajagopalan and Schulman extended the two-party case and considered a network of n parties that wish to compute some function of their private inputs by communicating over an arbitrary¹ noisy network. The work of [27] shows that if each channel is the binary symmetric channel² (BSC_ϵ), then one can obtain a *coding scheme* that takes any protocol Π that assumes noiseless communication, and converts it into a resilient protocol that computes the same task over the noisy network.

The coding scheme in [27] defies noise by adding redundancy. The amount of added redundancy is usually measured with respect to the noiseless setting—the *rate* of the coding scheme is the communication of the noiseless protocol divided by the communication of the noise-resilient one. The rate assumes values between zero and one, and ideally is bounded away from zero, commonly known as *constant* or *positive* rate. The rate may vary according to the network in consideration, for instance, the rate in [27] behaves as $1/O(\log(d+1))$ where d is the maximal degree in the network. Hence, for networks where the maximal degree is non-constant, the rate approaches zero as the network size increases.

The next major step for multiparty coding schemes was provided by Jain, Kalai, and Lewko [24] and by Hoza and Schulman [23]. In these works the noise is no longer assumed to be stochastic but instead is adversarial. That is, they consider worst-case noise where the only limit is the number of bits flipped by the adversary. They showed that as long as the adversary flips at most ϵ/m -fraction of the total communication, a coding scheme with a constant rate can be achieved, where ϵ is some small constant, and m is the number of communication links in the network.³

While both these works consider adversarial errors, they consider different communication models. The work [23] assumes that every party sends a single bit to all its neighbors in each round;

¹By “arbitrary” we mean that the topology of the network can be an arbitrary graph $G = (V, E)$ where each node is a party and each edge is a communication channel connecting the parties associated with these nodes.

²That is, a channel that flips every bit with some constant probability $\epsilon \in (0, 1/2)$.

³[24] obtained this result for the specific star network, whereas [23] generalized this result to a network with arbitrary topology.

scheme	topology	noise level	noise type	rate	efficient
RS94 [27]	arbitrary		BSC_ϵ	$1/O(\log(d+1))$	No
ABEGH16 [3]	clique		BSC_ϵ	$\Theta(1)$	Yes
HS16 [23]	arbitrary	$O(1/m)$	substitution	$\Theta(1)$	No
HS16 [23]	arbitrary	$O(1/n)$	substitution	$1/O(m \log(n)/n)$	No
JKL15 [24]	star	$O(1/m)$	substitution	$\Theta(1)$	No
Algorithm A	arbitrary	$O(1/m)$	oblivious insertions and deletions	$\Theta(1)$	Yes
Algorithm B	arbitrary	$O(1/(m \log m))$	insertions and deletions	$\Theta(1)$	Yes

Table 1: Interactive coding schemes in the multiparty setting. The first four are in the *fully-utilized* model, while JKL15 [24] and all algorithms in this paper are in the *non-fully-utilized* model.

this model is sometimes called *fully utilized*, and is commonly considered in multiparty interactive coding [3, 8, 23, 27]. On the other hand, the work in [24] uses a relaxed communication setting, in which parties may or may not speak in a given round; this setting is very common for distributed computations, and is relatively less studied in previous work on interactive coding. Furthermore, in [24] the *underlying* protocol is not assumed to be fully utilized; instead this work uses the (weaker) assumption that the order of speaking in the underlying protocol is known, and is independent of the parties' inputs. Naturally, one can convert any protocol in the non-fully-utilized model to a fully-utilized protocol by forcing all parties to speak at every round, and then applying an interactive coding scheme to the fully-utilized protocol. However, the conversion to a fully-utilized protocol may cause the communication complexity to increase by a factor of up to m , greatly harming the rate of the coding scheme.

1.1 Our Contributions and Comparison to Previous Work

We design coding schemes for adversarial noise in the same communication model as [24]; namely, where parties may or not speak in a given round, and the underlying protocol is only assumed to have a fixed speaking order. One of the most interesting aspects of our work is that our coding schemes are *computationally efficient*, unlike either [24] or [23].⁴

Also, we consider a stronger type of noise, *insertions and deletions*, albeit in the synchronous setting. This type of noise may completely remove a transmission (so that the receiver is not aware that a bit was sent to him), or inject new transmissions (so that the receiver receives a bit while the sender didn't send anything). Insertion and deletion noise is more general, and considered to be more difficult, than bit-flips. Indeed, a bit flip (a *substitution noise*) can be simulated by a deletion followed by an insertion. We note that insertions and deletions are trivially correctable in the fully utilized communication model; indeed, each party expects to hear a message from each of its neighbors in each round, so a deletion reduces to an erasure. By contrast, in the non-fully-utilized setting, insertions and deletions seem non-trivial to correct. Indeed, as first noted by Hoza [22], it seems *crucial* to allow insertion and deletion errors in the non-fully-utilized setting to make the problem non-trivial. To see this, consider a party that speaks once every two rounds—on

an even round to communicate the bit '0' and on an odd round to communicate '1'. In fact, this communication is completely resilient to noise that only flips bits since only the timing of the transmission matters.

Finally, our coding scheme works on arbitrary topologies, in contrast to [24], which only applies to the star graphs. As mentioned above, in the fully utilized model, [23] also handles arbitrary graphs.

Our first coding scheme (Algorithm A) achieves a constant rate, and is resilient to ϵ/m -fraction of adversarial insertion, deletion, and substitution noise.⁵ This scheme assumes a limited type of noise, i.e., an *oblivious adversary* whose corruptions are predetermined and are independent of parties' inputs and randomness. In our second coding scheme (Algorithm B), we remove the restriction of the obliviousness of the adversary, at the price of being resilient to $\epsilon/(m \log m)$ -fraction of errors. See Table 1 for a comparison of our new results (Algorithm A and Algorithm B) with the state of the art.

1.2 Future Directions

Before describing our results and techniques in further detail, we stop to highlight promising directions for future work. While we consider error rates of at most $O(1/m)$ in this paper, this is not the optimal noise rate for multiparty interactive coding in general. Indeed, Hoza and Schulman [23] further improve their noise resilience to $\Omega(1/n)$, where n is the number of parties, at the expense of reducing the rate to $1/O(m \log(n)/n)$, which is no longer constant. It is an interesting open question whether there exists a constant-rate coding scheme with noise resilience $\Omega(1/n)$, in any communication model. Furthermore, we note that our noise resilience to non-oblivious adversaries is $O(1/m \log(m))$, as opposed to the $\Omega(1/m)$ noise resilience in [23] and [24]; this seems to be inherent to our approach, designed to yield computational efficiency. We discuss this barrier further in Section 3.

Even though our noise-resilient protocol increases the communication complexity by only a constant factor, it may blow up the *number of rounds* of communication by more than a constant factor. In our model, unlike the fully utilized model, the communication complexity does not determine the round complexity. Specifically, in our model an interactive protocol with communication complexity CC may consist of CC/m rounds (in the case that the network is fully utilized) or may consist of CC rounds (in the case where

⁴These coding schemes utilized a combinatorial object known a *tree code*, for which no efficient construction is known.

⁵We assume that each insertion, deletion, or substitution counts as a single corruption.

the communication is very sparse). Simulating either one of these interactive protocols with the algorithms in this paper will take $\Theta(CC)$ rounds. It is an interesting question as to whether one can extend our results to simultaneously achieve constant blowup in rounds.

2 MAIN RESULTS

We give efficient interactive coding schemes with constant rate for arbitrary synchronous networks (not necessarily fully-utilized) that suffer from a certain fraction of insertion, deletion and substitution noise. Our result is two-fold. First, we assume that the adversary is oblivious, i.e., the noise is predetermined and is independent of the randomness used by the parties.

THEOREM 2.1 (OBLIVIOUS NOISE, INFORMAL). *Let $G = (V, E)$ be an arbitrary synchronous network with $n = |V|$ nodes and $m = |E|$ links. For any noiseless protocol Π over G with a predetermined order of speaking, and for any sufficiently small constant ϵ , there exists an efficient coding scheme that simulates Π over a noisy network G . The simulated protocol is robust to adversarial insertion, deletion, and substitution noise, assuming at most ϵ/m -fraction of the communication is corrupted. The simulated protocol communicates $O(CC(\Pi))$ bits, and succeeds with probability at least $1 - \exp(-CC(\Pi)/m)$, assuming the noise is oblivious.*

As an intermediate step towards proving Theorem 2.1, we first prove the result assuming that the parties pre-share a common random string (to which the adversary is oblivious). Then we show how to remove this assumption without any loss in parameters. We give more details in Section 4.

Next, we remove the restriction that the noise is oblivious. Namely, we consider adversaries that may adaptively decide which transmissions to corrupt according to the observed transcript, as well as the parties inputs (however, the noise is unaware of any private coin-tossing a party may perform later in the protocol). In this case we still obtain an efficient coding scheme with a constant rate, albeit, with slightly smaller noise resilience.

THEOREM 2.2 (NON-OBLIVIOUS NOISE, INFORMAL). *Let $G = (V, E)$ be an arbitrary synchronous network with $n = |V|$ nodes and $m = |E|$ links. For any noiseless protocol Π over G with a predetermined order of speaking, and for any sufficiently small constant ϵ , there exists an efficient coding scheme that simulates Π over a noisy network G . The simulated protocol is robust to adversarial insertion, deletion, and substitution noise, assuming at most $(\epsilon/m \log m)$ -fraction of the communication is corrupted. The simulated protocol communicates $O(CC(\Pi))$ bits, and succeeds with probability at least $1 - \exp(-CC(\Pi)/m)$.*

3 THE CODING SCHEME

3.1 The multiparty interactive-communication model

Before proceeding to describe our coding scheme, we elaborate on our communication model. The communication model works in synchronous rounds as follows. At each round, any subset of parties may decide to speak. Each link is allowed to transmit at most one symbol per round in each direction from a constant-sized alphabet Σ . We will assume throughout this paper that $\Sigma = \{0, 1\}$ (both in the

noiseless and noisy settings), however, our results extend to a larger alphabet as well. At each round, a party is allowed to send multiple (possibly different) symbols over multiple links. A *transmission* over a certain link at a certain round is the event of a party sending a message on this link at that round (if both parties send messages these are two separate transmissions).

We re-emphasize that, contrary to most previous work, our communication model is not fully-utilized and does not demand all parties to speak at each round on every communication channel connected to them; in fact we don't demand a certain party to speak at all at any given round.

3.2 Towards a robust simulation: Key ideas

In this section we motivate the elements of our coding scheme at a high level. The basic idea towards constructing a multiparty coding scheme is to have each pair of parties perform a two-party coding scheme [23, 24, 27]. However, merely correcting errors in a pairwise manner is insufficient, since if a pair of parties found an inconsistency and backtracked, this may cause new inconsistencies (in particular, between these parties and their other neighbors). In [24], this problem was solved by assuming there is one party who is connected to all parties (i.e., the star topology). This party has a global view on the progress of the simulation in the *entire network*, and hence can function as the “conductor of the orchestra.”

In our setting, no such central party exists and consequently no party knows the state of the simulation in the entire network. Instead, each party only sees its local neighborhood, and needs to propagate its “status” to the entire network in a completely decentralized manner.

We mention that Hoza and Schulman [23] also considered an arbitrary topology, but they consider the fully-utilized model. Correcting errors efficiently in the non-fully-utilized model seems to be trickier, since errors may remain unnoticed longer just due to the fact that parties may communicate very rarely. We elaborate on this towards the end of this section.

In order to keep our simulation *efficient*, as opposed to previous works in the multiparty setting which used the (inefficient) tree-code approach, we use the rewind-if-error approach [5, 14, 20, 25, 28] (see also [13]). Namely, each two neighboring parties send a hash of their simulated pairwise transcripts, and if the hashes do not match, then an error is detected, and the two parties initiate a “meeting-points” mechanism [20, 28] in order to find a previous point where their transcripts agree. Since we want a constant-rate coding scheme with error rate up to $\Omega(1/m)$, our goal is to ensure that any corruption only causes the network to waste $O(m)$ communication. Indeed, if the parties need K communication to correct a single error, then constant-rate coding implies a maximal noise level of $1/K$. If the noise level is higher, the communication required to correct the errors is already too high to preserve a constant-rate coding. Therefore, to tolerate error rate $\Omega(1/m)$, in our coding schemes the parties will have to detect errors by communicating at most $O(1)$ bits per check, so that the consistency-checking costs $O(m)$ communication overall. In hindsight, this will restrict the parties to use hash functions with *constant-sized outputs*.

Let us now describe a preliminary attempt at a coding scheme. The coding scheme will have many *iterations*, where each iteration

will consist of the following. First, every pair of parties will send each other hashes of their simulated pairwise transcripts. If these match, they continue simulating with each other for a small number of rounds; else, they use the meeting points mechanism to find a common “meeting” point where both their partial transcripts agree and rewind their partial transcript back to that point. Until they have done so, they refuse to simulate with any other party. The parties continue repeating these two steps (1. Send Hashes / Do Meeting Points and 2. Simulate / Stay Silent) until they finish simulating the underlying protocol. Each step will take a fixed number of rounds, and since we are in the synchronous model, the parties will always be able to tell which part currently is being carried out.

This naive coding scheme has a clear but important flaw, which was alluded to at the beginning of this subsection. Namely, once a party u decides to rewind due to an error on the link (u, v) , this has an effect on the simulation of u with its other neighbors. In order to ensure correctness of the overall simulation, u must rewind the simulation with each and every one of its neighbors that may be affected.

It is tempting to rewind u 's neighborhood by having u immediately truncate its simulated transcript for each adjacent link (u, w) , to the same place it rewound the link (u, v) . After such a truncation, a discrepancy would appear on the link (u, w) since w did not change its simulated transaction, and the meeting points mechanism would resolve any such resulting discrepancy in the next iteration. In particular, an error on the link (u, v) would cause u and v to perform meeting points and rewind their transcript to a consistent point. Then, in the next iteration, u will be inconsistent with its other neighbors and will initiate meeting points with all its other neighbors, resulting in them rewinding their transcripts accordingly. In the following iteration, any neighbor w of u would find an inconsistency with any other party in its *own* neighborhood, initiate meeting points with them, and so on until the entire network has rewound.

While the entire network eventually rewinds in this scheme, the number of iterations this takes is proportional to the diameter of the network. Recall that to ensure constant rate, we need to correct any error with at most $O(m)$ communication (with high probability), which in our case means just $O(1)$ iterations. So this naive approach could *not* achieve constant rate.

Worse still, even if we augmented the consistency check step so that news propagates through the entire network in just a single iteration, this approach would still fail with high probability. The consistency checking involves exchanging hashes of transcripts and verifying whether they match. *Hash collisions* are a major problem here; a hash collision will cause parties u and v to believe that their transcripts are consistent, when really they are not. This can cause u and v to continue simulating when they should not, wasting communication on an incorrect simulation that will have to be rewound later. Since we are restricted to use constant-sized hashes to keep the rate constant, hash collisions occur with constant probability in each exchange. Note that at least $n - 1$ hash exchanges are required to inform all n parties that an error has occurred. The probability that none of these hash exchanges suffers a collision is negligible, hence, many parties will not be informed of the error and will not rewind their transcript.

What saves us here is that u does not need to initiate a meeting points protocol to tell w to rewind— u can tell w to rewind *directly*. The meeting points protocol is designed to find a point where the transcripts of u and w are consistent, when neither u nor w has any idea where that point may be. But in the above case, u is confident that w should rewind, and should simply inform w that it should rewind.

To summarize the above ideas, each iteration in our coding scheme consists of the following phases: 1. Consistency Check, 2. Simulate / Stay Silent, and 3. Rewind request (if necessary). As before, each of these phases takes a fixed number of rounds, so the parties are always in sync with regards to which phase they are in.

Some care needs to be taken with the Rewind phase. Specifically, u may not be able to instantly communicate to its neighbor w how far it wants w to rewind. For instance, u may have rewound a very large number of rounds, and even telling its neighbor how many rounds to rewind would require a large amount of communication; this cannot be communicated in the fixed number of rounds that the rewind phase contains. Instead, u should be satisfied with just telling w to rewind a small amount at a time, spanning the rewind process over several consecutive iterations.

We stress that in the multiparty setting, hash collisions are very expensive (that is, they cause the parties to waste a lot of communication), and so the parties should try to minimize the potential number of hash collisions. For instance, in the above process of rewinding w , party u keeps sending hashes according to the partially rewound transcript. This would make the link (u, w) look “consistent” to the two parties, since both truncated the same amount of chunks each time. This prevents u and w from initiating a meeting points protocol on an inconsistent link, thus reducing the potential of more hash collisions, due to the fact that when the link is consistent, no hash collision can occur. This is one of the key conceptual observations behind our work.

There is one final issue that is specific to the non-fully-utilized model. Recall that in the non-fully-utilized model, communication may be sparse. For concreteness, assume the following sparse protocol over the *line* (i.e., a path graph, where party $i \in \{1, 2, \dots, n-1\}$ is connected to party $i+1$). In this example protocol, party 1 communicates back-and-forth with party 2 for n rounds while other parties remain silent, then party 2 communicates with party 3 for n rounds while other parties remain silent, and so on, until the communication reaches parties $n-1$ and n , and bounces back towards party 1. The communication that occurs in any set of n rounds of this underlying protocol is n , since only a single party is speaking in any given round. An iteration of our coding scheme would give the parties n rounds to simulate, after which they perform a single consistency check.

However, when *simulating* the underlying protocol in our coding scheme, the parties may not all be in agreement about which round they are simulating. For example, parties 1, 2, and 3 could think they are simulating round 1 (where parties 1 and 2 talk a lot), while party 4, 5, and 6 think they are simulating round $3n+1$ (where parties 4 and 5 talk a lot), and generally parties $3i+1, 3i+2$, and $3i+3$ think they are simulating round $i \cdot n + 1$. In this case, there will be $\Omega(n^2)$ communication in just one iteration of our coding scheme (in $O(n)$ rounds). This is potentially disastrous. Not only might this

lead to the rate of the coding scheme being super-constant, but also the adversary would have the budget to place $\Omega(\epsilon n)$ additional errors, and may be able to use these errors to derail any attempts of getting the simulation back on track.

To avoid this issue altogether, we introduce a “flag-passing” phase in which each party informs the entire *network* whether all seems correct and the simulation should continue, or if it sees an inconsistency and the network should idle while this is fixed. This fixes the issue above, since party 3 will realize that there is an inconsistency between its transcripts with parties 2 and 4, and will notify the network to avoid simulation.

Putting it all together, our resilient protocol consists of repeatedly executing the following four phases in order: (i) consistency check, (ii) flag passing, (iii) simulation, and (iv) rewind. The coding scheme cycles through these four phases in a fixed manner, where each phase takes a fixed amount of rounds to avoid ambiguity. We now describe these phases in thorough detail.

3.3 The Basic Coding Scheme

Before we describe our scheme, let us elaborate on why we need to assume that in the underlying (noiseless) protocol Π , the speaking order of the parties does not depend on the parties’ inputs or communication so far. This (limiting) assumption is crucial to keep our coding scheme with a constant rate despite $\Theta(1/m)$ fraction of noise. To see that, consider the simulation phase, and assume that it consists of a fixed number of rounds, say, K rounds. Now, the communication in each phase may be between K and Km bits. On one hand, the communication cannot be too small: the other phases take $\Theta(m)$ communication, hence we need $K = \Omega(m)$ to keep the rate constant. On the other hand, the communication cannot be too large. Assume $K = m$ and the communication is m^2 . This means that a single error would rewind the protocol by causing the retransmission of another m^2 bits. As mentioned above, wasting $\omega(m)$ bits per error would prevent us from reaching $\Theta(1/m)$ resilience.

Our solution to the above conundrum is to maintain a fixed amount of *communication* in each simulation phase. We will define a *chunk* as a set of consecutive rounds that consists of $\Theta(m)$ communication in our underlying protocol Π ; note that different chunks may have different number of rounds, but they all have the same amount of bit transmissions. In order to split the underlying protocol into chunks, its communication must be known at the beginning of the coding scheme—it must be fixed and independent of inputs and noise.

While for Algorithm A (the oblivious-noise case) we need each chunk to contain $\Theta(m)$ bits of communication, in Algorithm B (the non-oblivious case) we will actually need it to be $\Theta(m \log(m))$. Planning ahead, we will set the amount of communication in a chunk via the parameter K , and require that the communication in any chunk will be between $4K$ and $5K$. We assume for now that $K = m$, and later (in Algorithm B) increase it to $K = m \log m$. We let $|\Pi|$ denote the number of chunks in Π .

We now summarize the different phases and provide pseudo-code.

- (i) **Consistency check:** Each time the parties enter the “consistency check” phase they do a *single* iteration of the meeting-points mechanism [20], which consists of sending two hashes. If the hashes mismatch, they send the next two hash

values (of some prefixes of the transcript, as instructed by the meeting-points mechanism) *next time they enter the consistency check phase*. In each iteration the hash functions are seeded with fresh randomness, assumed to be shared between the two parties.

- (ii) **Flag passing:** Each party starts by generating a “flag” (i.e., a stop/continue bit) based on its state after the Consistency Check phase: if it experienced a hash mismatch with a neighbor, or has two neighbors with whom it has simulated up to different chunks in the underlying protocol, the flag is “stop,” and otherwise it is “continue.” It passes this flag along a fixed spanning tree $\mathcal{T}$ of G . Namely, each party receives flags from each of its children in $\mathcal{T}$. If one of the flags is stop, or if the party sees inconsistency with one of its neighbors, it sends a stop flag to its parent in the tree. Otherwise, it sends its parent the continue flag. After this phase ends and the root of $\mathcal{T}$ receives all the flags, the root propagates the computed flag in the opposite direction back to the leaves.
- (iii) **Simulation:** In this phase, the parties “execute” the next chunk of Π , sending and receiving messages as dictated by the protocol Π . This phase *always* takes $5K$ rounds, which is the maximal number of rounds required to simulate $5K$ transmissions of Π . The simulation of a specific chunk may take fewer rounds; in this case, the phase still takes $5K$ rounds where all the parties remain silent after the chunk’s simulation has completed until $5K$ rounds have passed. We note that some parties may be aware that the simulation so far contains errors that were not corrected yet, based on the previous phases. In the simulation phase, these parties will send a dummy message $\perp$ to their neighbors to indicate they are not simulating. Similarly, a party will refuse to simulate if its flag (obtained in the previous phase) is “stop”.
- (iv) **Rewind:** If the partial transcript between u and v consists of k chunks, then u will send a “rewind” message to any neighbor z for which the partial transcript between u and z contains more than k chunks. Once a party u sends a rewind message to a neighbor z , party z will truncate one chunk of the transcript that corresponds to the link (u, z) , and might then want to send rewind messages to its own neighbors, who may then want to send even more rewinds. The phase takes n rounds altogether. This ensures that this cascade of rewinds has enough time to go through the entire network. Note, however, that each link rewinds only a *single chunk* per rewind phase; if the inconsistency is larger the rewind process spans along several rewind phases (in consecutive iterations).

The formal coding scheme, assuming oblivious noise and a common shared randomness is depicted as Algorithms 1–3 below. The meeting points procedure is given in Algorithm 4. These serve as our base coding scheme. We obtain Theorems 2.1 and 2.2 by introducing small modifications to this base scheme, which we now describe.

3.4 Modifications for Algorithms A and B

To get Algorithms A and B from our basic scheme, we need two steps. First, we need to remove the common random string while

Algorithm 1 A noise-resilient simulation of Π (for party u)

Let $T_{u,v}$ denote the partial, pairwise transcript between u and v according to u , and let $|T_{u,v}|$ denote the number of *chunks* simulated so far in $T_{u,v}$.

```

1: INITIALIZESTATE()

2: for  $i = 1$  to  $100|\Pi|$  do
3:   for all  $v \in N(u)$  in parallel do ▷ meeting points
4:      $status_{u,v} \leftarrow \text{MEETINGPOINTS}(u, v, S_{i,u,v})$ 
5:    $minChunk \leftarrow \min_{v \in N(u)} |T_{u,v}|$ 
6:   if exists  $v$  such that  $status_{u,v} = \text{"meeting points"}$  then
7:      $status_u \leftarrow 0$ 
8:   else if exists  $v$  such that  $|T_{u,v}| > minChunk$  then
9:      $status_u \leftarrow 0$ 
10:  else
11:     $status_u \leftarrow 1$ 

12:  $netCorrect_u \leftarrow \text{FLAGPASSING}(u, status_u)$  ▷ flag passing

13: if  $netCorrect_u = 1$  then ▷ simulation
14:   Listen for one round.
15:   Simulate chunk  $|T_{u,v}| + 1$  with each party  $v \in N(u)$ 
   from whom we have not received  $\perp$  at the first round.
   The simulation is based on the partial transcript  $T_{u,w}$ 
   for each  $w \in N(u)$ , as well as the input to  $u$ .
16:   If the above step took less than  $5K$  rounds, wait until
    $5K$  rounds have passed
17:   if received no  $\perp$ 's in Line 14 in this iteration then
18:      $minChunk \leftarrow minChunk + 1$ 
19: else
20:   Send a single  $\perp$  to each neighbor, and wait  $5K$  rounds.

21: for round  $r = 1$  to  $n$  do ▷ rewind
22:   for all  $v \in N(u)$  in parallel do
23:     if  $status_{u,v} \neq \text{"meeting points"}$ 
   and  $alreadyRewound_{u,v} = 0$  then
24:       if  $|T_{u,v}| > minChunk$  then
25:         Send a rewind message to  $v$  and truncate
          $T_{u,v}$  by one chunk
26:          $alreadyRewound_{u,v} \leftarrow 1$ 
27:       if a rewind message is received from  $v$  then
28:         if  $status_{u,v} \neq \text{"meeting points"}$ 
   and  $alreadyRewound_{u,v} = 0$  then
29:           Truncate  $T_{u,v}$  by one chunk
30:            $alreadyRewound_{u,v} \leftarrow 1$ 

```

keeping an oblivious adversary, which gets us Algorithm A. Then, we make the protocol resilient to non-oblivious adversaries to get Algorithm B.

The basic idea for removing the common random string is adopted from previous work [5, 17, 20]: simply have each pair of parties sample this string and send it across their shared link. However, this

Algorithm 2 InitializeState()

```

1:  $K \leftarrow m$ 
2: for all neighbors  $v \in N(u)$  do
3:   Initialize  $T_{u,v} = \emptyset$ 
4:   Initialize  $k_{u,v}, E_{u,v}, mpc1_{u,v}, mpc2_{u,v} \leftarrow 0$ 
5:    $status_{u,v} \leftarrow \text{"simulate"}$ 
6:    $alreadyRewound_{u,v} \leftarrow 0$ 
7:    $S_{u,v} := (S_{i,u,v})_{i \in [100|\Pi|]} \xleftarrow{\text{unif}} (\{0, 1\}^{\Theta(|\Pi|K)})^{100|\Pi|}$ 
   (These are uniform bits of randomness assumed to be pre-
   shared between  $u$  and  $v$ .)
8:  $status_u \leftarrow 1$ .
9:  $netCorrect_u \leftarrow 1$ 

```

Algorithm 3 FlagPassing($u, status$)

Let $\rho \in V$ be a specific node known by all the parties. Let $\mathcal{T}$ be a spanning tree generated by a breadth-first-search starting from ρ . Denote the depth of $\mathcal{T}$ as $d(\mathcal{T})$, where the depth of a single vertex is 1. Finally, let the *level* of a vertex be defined as $\ell(v) := \ell(u) + 1$, where u is the parent of v in $\mathcal{T}$, and $\ell(\rho) = 1$.

```

1:  $netCorrect \leftarrow status$ 
2: if  $u$  is a leaf in  $\mathcal{T}$  then
3:   Send  $netCorrect$  to parent vertex in  $\mathcal{T}$ .
4:   Sleep for  $\ell(u) - 1$  rounds.
5: else
6:   Sleep for  $d(\mathcal{T}) - \ell(u)$  rounds. Ignore any messages received
   in these rounds.
7:   Receive  $b_1, \dots, b_k$ , one symbol from each child in  $\mathcal{T}$ .
8:    $netCorrect \leftarrow \bigwedge_{i=1}^k b_i \wedge status$ 
9:   if  $u \neq \rho$  then
10:    Send  $netCorrect$  to parent.
11:    Sleep for  $\ell(u) - 1$  rounds. Ignore any messages received
   in these rounds.
12:   if  $u = \rho$  then
13:    Send  $netCorrect$  to children.
14:   else
15:    Sleep for  $\ell(u) - 1$  rounds. Ignore any messages received in
   these rounds.
16:    Receive  $b$  from parent.
17:     $netCorrect \leftarrow b \wedge status$ 
18:    if  $u$  is not a leaf in  $\mathcal{T}$  then
19:      Send  $netCorrect$  to children.
20:      Sleep for  $d(\mathcal{T}) - \ell(u)$  rounds. Ignore any messages re-
   ceived in these rounds.
21: return  $netCorrect$ 

```

random string is very long, and sending it will increase the communication complexity by a quadratic factor. Therefore, instead, the parties communicate a short random string S which serves as a seed that generates a longer δ -biased string. This is done via a well-known technique by Naor and Naor [26], or Alon et al. [4]. This δ -biased randomness is used instead of the shared randomness

Algorithm 4 MeetingPoints($u, v, S_{i,u,v}, K$)

```

1: Method called by  $u$ , with  $v \in N(u)$ .  $S_{i,u,v} \in \{0, 1\}^{\Theta(|\Pi|K)}$  is a
   large random seed to be split up and used for hashing.  $\Pi \leftarrow$ 
   protocol to be simulated.
2:  $h \leftarrow$  inner product hash family w/ input length6  $\Theta(|\Pi|K)$ ,  $p =$ 
 $2^{-\Theta(K/m)}$  sufficiently small,  $o = \Theta(K/m)$ ,  $s = |S_{i,u,v}|/10$ .
3:  $(S_1, S_2, \dots, S_{10}) \leftarrow S_{i,u,v}$ .  $S_{i,u,v}$  is split into ten seeds. Wlog
   we assume that  $S_1, \dots, S_5$  are for the hashes it sends, and
 $S_6, \dots, S_{10}$  are for the hashes it uses for comparisons (so  $v$ 
   is using  $S_6, \dots, S_{10}$  for the hashes it sends, and  $S_1, \dots, S_5$  for
   comparisons, respectively).
4:  $k \leftarrow k + 1$ 
5:  $\tilde{k} \leftarrow 2^{\lfloor \log k \rfloor}$ . Let  $c$  be the largest integer such that  $c\tilde{k} \leq |T_{u,v}|$ .
6:  $T_1 \leftarrow T_{u,v}[1 : c\tilde{k}]$ ,  $T_2 \leftarrow T_{u,v}[(c-1)\tilde{k} + 1 : c\tilde{k}]$ 
7: Send  $(h_{S_1}(k), h_{S_2}(T_1), h_{S_3}(T_1), h_{S_4}(T_2), h_{S_5}(T_2))$  to neighbor  $v$ .
8:  $(H_k, H_{T_1}^{(1)}, H_{T_2}^{(1)}, H_{T_1}^{(2)}, H_{T_2}^{(2)}) \leftarrow$ 
 $(h_{S_6}(k), h_{S_7}(T_1), h_{S_8}(T_2), h_{S_9}(T_1), h_{S_{10}}(T_2))$ 
9: Receive  $(H'_k, H_{T_1}^{(1)'}, H_{T_1}^{(2)'}, H_{T_2}^{(1)'}, H_{T_2}^{(2)'})$  from our neighbor  $v$ .
10: if  $H_k \neq H'_k$  then
11:    $E \leftarrow E + 1$ 
12: if  $k = 1, E = 0$ , and  $H_{T_1}^{(1)} = H_{T_1}^{(1)'}$  then
13:    $k \leftarrow 0$ 
14:    $status \leftarrow$  “simulate” return  $status$ 
15: if  $H_{T_1}^{(1)} = H_{T_1}^{(1)'}$  or  $H_{T_1}^{(2)} = H_{T_1}^{(2)'}$  then
16:    $mpc1 \leftarrow mpc1 + 1$ 
17: else if  $H_{T_2}^{(1)} = H_{T_2}^{(1)'}$  or  $H_{T_2}^{(2)} = H_{T_2}^{(2)'}$  then
18:    $mpc2 \leftarrow mpc2 + 1$ 
19: if  $2E \geq k$  then
20:    $k \leftarrow 0, E \leftarrow 0, mpc1 \leftarrow 0, mpc2 \leftarrow 0$ 
21:    $status \leftarrow$  “meeting points”
22: else if  $k = \tilde{k}$  then
23:   if  $mpc1 > 0.4k$  then
24:      $T \leftarrow T_1$ 
25:      $k \leftarrow 0, E \leftarrow 0$ 
26:   else if  $mpc2 > 0.4k$  then
27:      $T \leftarrow T_2$ 
28:      $k \leftarrow 0, E \leftarrow 0$ 
29:    $mpc1 \leftarrow 0, mpc2 \leftarrow 0$ 
30:    $status \leftarrow$  “meeting points”
31: else
32:    $status \leftarrow$  “meeting points”
33: return  $status$ 

```

in Algorithm 1, and this yields Algorithm A. Specifically, Algorithm A uses Algorithm 5 for InitializeState(), which in turn uses Algorithm 6 to share the δ -biased randomness.

To get Algorithm B from Algorithm A, all we need to do is change the chunk size K to be $m \log(m)$ instead of m . Specifically, Algorithm B uses Algorithm 7 for InitializeState(). We provide an overview of the analysis of Algorithms A and B in Section 4.

Algorithm 5 InitializeState() (for simulation without assuming a CRS)

```

1:  $K \leftarrow m$ 
2: for all neighbors  $v \in N(u)$  in parallel do
3:   Initialize  $T_{u,v} = \emptyset$ 
4:   Initialize  $k_{u,v}, E_{u,v}, mpc1_{u,v}, mpc2_{u,v} \leftarrow 0$ 
5:    $status_{u,v} \leftarrow$  “simulate”
6:    $alreadyRewound_{u,v} \leftarrow 0$ 
7:    $\delta := 2^{-\Theta(|\Pi|K/m)}$ 
8:    $S := (S_{i,u,v})_{i \in [100|\Pi|]} \leftarrow \text{RandomnessExchange}(u, v, \delta, K)$ 
9:    $status_u \leftarrow 1$ 
10:   $netCorrect_u \leftarrow 1$ 

```

Algorithm 6 RandomnessExchange(u, v, δ, K)

input: u calling party, $v \in N(u)$. $\delta > 0$ is a parameter that specifies the bias we desire from our random string.

```

1: Fix an arbitrary total order on the vertices in  $V$  (common for
   all instantiations of this procedure).
2:  $\ell \leftarrow \Theta(|\Pi|K \cdot 100|\Pi|)$ .
3:  $r \leftarrow \Theta(\log(1/\delta) + \log(\ell))$ , large enough to generate a string of
   length at least  $\ell$  and bias at most  $\delta$ .
4:  $C \leftarrow$  error correcting code with block length  $\Theta(r)$ , constant
   rate and constant distance.
5: if  $u < v$  in the ordering then
6:   Sample a uniformly random string  $L \in \{0, 1\}^r$ 
7:   Send  $C(L)$  to  $v$ 
8: else
9:   Receive  $W$  from  $v$ 
10:   $L \leftarrow C^{-1}(W)$ , the decoding of  $W$  with respect to the code  $C$ .
11:  $S_{u,v} \leftarrow \left( \{0, 1\}^{\Theta(|\Pi|K)} \right)^{100|\Pi|}$  gets the  $\delta$ -biased string of
   length  $\ell$  generated from  $L$ .
12: Output  $S_{u,v}$ .

```

Algorithm 7 InitializeState() for nonoblivious noise

```

1:  $K \leftarrow m \log(m)$ 
2: for all neighbors  $v \in N(u)$  in parallel do
3:   Initialize  $T_{u,v} = \emptyset$ 
4:   Initialize  $k_{u,v}, E_{u,v}, mpc1_{u,v}, mpc2_{u,v} \leftarrow 0$ 
5:    $status_{u,v} \leftarrow$  “simulate”
6:    $alreadyRewound_{u,v} \leftarrow 0$ 
7:    $\delta := 2^{-\Theta(|\Pi|K/m)}$ 
8:    $S := (S_{i,u,v})_{i \in [100|\Pi|]} \leftarrow \text{RandomnessExchange}(u, v, \delta, K)$ 
9:    $status_u \leftarrow 1$ 
10:   $netCorrect_u \leftarrow 1$ 

```

4 ANALYSIS

A main hurdle is analyzing the correctness of the scheme and its probability of success. Similar to previous work, our analysis utilizes a potential argument that measures the progress of the simulation. The progress is measured in two parts. The first part is the progress of simulating the partial transcript of each individual link. This

part is similar to the two-party case [20]. The second is the “global” progress which considers the progress of the simulation in the entire network. Each part entails its own potential function; this ‘divide-and-conquer’ approach greatly simplifies the analysis.

By defining a potential function in this manner, we can show that the parties make significant progress towards simulating the noiseless protocol correctly in each iteration as long as the adversary does not commit too many errors and there are not too many hash collisions. Ensuring that these bad events do not occur is the main technical difficulty of the analysis.

The first difficulty is to bound the number of errors the adversary can commit, and this arises from the fact our communication model is not fully-utilized. Namely, recall that the adversary’s error budget is proportional to the communication of the parties. Knowing this, the adversary may try to insert errors that will cause the parties get out of sync, and as a result speak all simultaneously, which in turn gives the adversary more budget for errors. To rule this out, we show that the communication in any iteration of our coding scheme is proportional to $O(m(1 + \ell))$, where ℓ is the number of errors or hash collisions that have occurred in that iteration. Recalling that an iteration of our coding scheme simulates a chunk consisting of m bits of communication, this lets us prove that the communication overall in our coding scheme is $O(CC(\Pi))$ with high probability over the hashes. This analysis crucially relies on the flag-passing step, where parties keep silent whenever they suspect an error has occurred, rather than continue in the simulation.

The second difficulty relates to bounding the number of hash collisions when the collision probability is a constant. We note that in the two-party coding scheme of [20] hash collisions also occur with constant probability. Just like in that work, we want to bound the number of hash collisions that occur, even when the parties do not use a CRS and the adversary is non-oblivious. However, we encounter significant obstacles when extending the same approach to the multiparty setting, which stem from the fact that we are dealing with a non-fully-utilized multiparty protocol, which gives the adversary more flexibility in their attacks.

Our analysis is performed in three steps. We begin by assuming the parties share a common random string (CRS), and assume that the noise is independent of this CRS. The random string is used as seeds to the hash functions. Since the CRS is independent of the noise, we can bound (with high probability) the number of hash collisions throughout the coding scheme. This in turn bounds the increase in the total communication attributed to hash collisions, by a constant factor. This part of the analysis is similar to the analysis done in previous works, e.g., [20].

The next step is removing the CRS. We replace the CRS with a δ -biased string, and argue that the number of hash collisions does not increase by much, assuming the adversary is oblivious to the δ -biased string. The final step is considering the case where the adversary is non-oblivious. Here we union-bound over all the possible oblivious attacks, yet, even here we must address an obstacle that does not show up in the two-party setting. Let us expand on the last two steps of our analysis.

4.1 Removing the Common Random String

As mentioned earlier, the idea for removing the common random string is to have each pair of parties agree on a short random string, and use this as a seed to generate a longer δ -biased string that they use for randomness in the protocol. The key intuition as to why δ -biased randomness suffices is that hashing with a δ -biased seed behaves “close” to hashing with a truly uniform seed; this idea dates back to the papers of Naor and Naor [26] and Alon et al. [4]. However, this holds only when the input to the hash function is independent of the δ -biased seed. Unfortunately, in our setting, the input to the hash function may depend on the δ -biased seed: indeed, the parties hash their partial transcripts in each meeting points phase, and these partial transcripts *do* depend on seed, since they are a function of the output of previous hashes, e.g., on whether or not hash collisions occurred in previous rounds.

We circumvent this issue by showing that a given pattern of hash collisions determines the (partial) transcripts throughout the protocol. Once the transcripts are fixed, the hashes output behave similarly to the case of uniform randomness, up to a statistical difference of at most δ . Therefore, if δ is small enough, we can prove robustness for all possible fixed patterns of hash collisions. Then, we can union bound over all the possible patterns of hash collisions, which bounds the failure probability for the δ -biased case by the failure of the uniform case plus an error term that depends on δ and the cardinality of the different hash patterns. One cannot take δ to be too small, since generating the δ -biased string requires communicating $\approx O(m \log(1/\delta))$ bits, hence the smaller δ gets, the more communication we need. Luckily, we can set δ to be sufficiently small so that the probability of failure is still exponentially small even after the union bound, while keeping a constant rate in the communication.

Finally, we note that sharing the seeds for δ -biased randomness requires communication, so the adversary may try to corrupt this communication and mess up the “shared” randomness for some pair of parties. Just like in the work of [20], we handle this by encoding the seed using an error correcting code with sufficiently large distance. However, in the non-fully-utilized setting, an adversary might try to corrupt the randomness exchange in the hopes that the increased communication in the rest of the protocol will pay for their errors. We address this similarly to how we bound the communication for Algorithm 1 (the basic coding scheme). Namely, we show that corrupting the randomness requires so many errors that the adversary can only do so by exceeding their budget.

4.2 Dealing with a non-oblivious adversary

Finally, we consider the case where the adversary is non-oblivious. We would like to follow the approach of [20] and simply union bound over all the possible oblivious attacks. Unfortunately, as opposed to the two-party case of [20], knowledge of the hash seeds gives the adversary too many options for attacks, and the proof fails spectacularly.

To see why this is, fix an edge (u, v) in the network, and suppose for simplicity that we use truly random functions for our hashes. Since we use constant length hash outputs and the adversary knows the random seeds ahead of time, with constant probability over the random seed, the adversary can corrupt the simulation on

the link (u, v) such that a hash collision is guaranteed in the next meeting points phase. In this case, the parties will continue to simulate (incorrectly). But there is no guarantee that it will be caught in the next phase either; if the adversary was fortunate, perhaps he was able to make an error that causes hash collisions in *two consecutive meeting points phases*. Generalizing this reasoning, with probability $1/m$, the adversary forces hash collisions on the link (u, v) for $\Theta(\log m)$ consecutive phases, leading to $\Theta(m \log m)$ wasted communication. Since there are m links for the adversary to choose from when making his error, with high probability there exists a link on which such an attack is possible. Therefore, he can create $\Theta(m \log m)$ wasted communication with just a single error.

To overcome this issue, we increase the length of the hash output, so it is no longer constant but rather $\Theta(\log m)$. Then the hash collision probability drops to $1/\text{poly}(m)$, and the union bound yields the desired outcome. However, the length of each hash increased from a constant to $\Theta(\log m)$, and the rate of the coding scheme is no longer constant. To fix this, we simulate the protocol in larger chunks. That is, instead of sending a hash every $\Theta(m)$ bits of communication of Π , parties exchange hashes of length $\Theta(\log m)$ every $\Theta(m \log m)$ bits of simulation, thus increasing the overall communication by just a constant factor.

5 RELATED WORK

As mentioned above, interactive coding was initiated by Schulman [28, 29]. Over the last several years there has been tremendous amount of work on interactive coding schemes in the two-party setting (e.g., [6, 7, 10, 14, 18, 19]), and in the multi-party setting (detailed below). We refer the reader to [13] for a survey on the field (and to references therein).

Coding schemes for insertions and deletions in the two-party setting were first constructed by Braverman, Gelles, Mao, and Ostrovsky [9]. As was noted above, in the model where in each round each party sends a single bit (which is the model used by most previous works, including [9]), insertions and deletions are only meaningful in the *asynchronous* model, as otherwise, such an error model is equivalent to the erasure model. However, the asynchronous model seems to be incompatible with insertions and deletions, since a single deletion can cause a “deadlock”, where both parties wait to the next incoming message. Braverman et al. suggested a model where any deletion is followed by an insertion, thus the protocol never “halts” due to noise. However, the noise may delete a certain message and then inject a spoofed “reply” to the original sender. Then, one party believes that the protocol has progressed by one step, while the other party is completely oblivious to this. This type of noise was called *synchronization attack* as it brings the parties out of synch.

Braverman et al. [9] constructed a coding scheme for insertions and deletions in this model with constant communication rate, and resilience to constant fraction of noise. Later, Sherstov and Wu [30] showed that a very similar scheme can actually resist an *optimal* noise level. Both these schemes are computationally inefficient. Haeupler, Shahrasbi, and Vitercik [21] constructed an efficient scheme that is resilient to (a small) constant fraction of insertions and deletions. Furthermore, they constructed a scheme where the communication rate approaches 1 as the noise level

approaches 0. Efremenko, Haramaty, and Kalai [12] considered the *synchronous* setting, where parties can send messages of arbitrary length in each round, and the adversary may insert and delete bits in the content of each message. They construct an efficient coding scheme with constant communication rate, and constant blowup in the round complexity, that is resilient to a constant fraction of noise. All the above schemes consider only the two-party setting.

In the multiparty setting, Rajagopalan and Schulman [27] constructed a coding scheme for stochastic noise that achieves a rate of $1/O(\log(d+1))$ for networks with maximal degree d . This implies a constant rate coding scheme for graphs with constant degree. Alon et al. [3] showed that if the topology is a clique, or a dense d -regular graph, then constant rate coding is also possible. Yet, Braverman, Efremenko, Gelles, and Haeupler [8] proved that a constant rate is impossible if the topology is a star. All the above works assume a synchronous fully-utilized network. Gelles and Kalai [15] showed that constant rate coding schemes are impossible also on graphs with constant degree, such as a cycle, assuming a synchronous, yet not fully-utilized model.

The case of adversarial noise in the multiparty setting was first considered by Jain, Kalai, and Lewko [24], who constructed a constant-rate coding scheme over an asynchronous star network that is resilient to $O(1/n)$ fraction of noise. Hoza and Schulman [23] considered synchronous networks with arbitrary topology and constructed a constant rate coding scheme that is resilient to $O(1/m)$ noise. Via routing and scheduling techniques, they show how to resist a fraction of $O(1/n)$ -noise, while reducing the rate to $1/O(m \log(n)/n)$. Both these schemes use tree-codes, and therefore are computationally inefficient.

Aggarwal, Dani, Hayes, and Saia [1] constructed an efficient synchronous coding scheme, assuming the parties have shared randomness that is unknown to the adversary. They consider a somewhat different communication model than the above, where the length of the protocol is not predetermined and may vary with the noise (similar to the two-party adaptive notion of [2]). Their coding scheme is resilient to an arbitrary (and *a priori* unknown) amount of bit-flips (as long as the noise-pattern is predetermined and independent of parties shared randomness), and has a rate of $1/O(\log(nCC(\Pi)))$.

Censor-Hillel, Gelles, and Haeupler [11] constructed asynchronous schemes, where the parties do not know the topology of the network (an assumption that is very common in the distributed computation community). Their scheme is resilient to $O(1/n)$ noise and has a rate of $1/O(n \log^2 n)$.

REFERENCES

- [1] Abhinav Aggarwal, Varsha Dani, Thomas P. Hayes, and Jared Saia. 2017. Distributed Computing with Channel Noise. Cryptology ePrint Archive, Report 2017/710. <https://eprint.iacr.org/2017/710>.
- [2] Shweta Agrawal, Ran Gelles, and Amit Sahai. 2016. Adaptive Protocols for Interactive Communication. In *2016 IEEE International Symposium on Information Theory (ISIT)*. 595–599. <https://doi.org/10.1109/ISIT.2016.7541368>
- [3] Noga Alon, Mark Braverman, Klim Efremenko, Ran Gelles, and Bernhard Haeupler. 2016. Reliable Communication over Highly Connected Noisy Networks. In *Proceedings of the 2016 ACM Symposium on Principles of Distributed Computing (PODC '16)*. 165–173. <https://doi.org/10.1145/2933057.2933085>
- [4] Noga Alon, Oded Goldreich, Johan Håstad, and René Peralta. 1992. Simple Constructions of Almost k -wise Independent Random Variables. *Random Structures & Algorithms* 3, 3 (1992), 289–304. <https://doi.org/10.1002/rsa.3240030308>

- [5] Zvika Brakerski and Yael Tauman Kalai. 2012. Efficient Interactive Coding against Adversarial Noise. *Proceedings of the IEEE Symposium on Foundations of Computer Science* (2012), 160–166.
- [6] Zvika Brakerski, Yael T. Kalai, and Moni Naor. 2014. Fast Interactive Coding Against Adversarial Noise. *J. ACM* 61, 6, Article 35 (Dec. 2014), 30 pages. <https://doi.org/10.1145/2661628>
- [7] M. Braverman and K. Efremenko. 2017. List and Unique Coding for Interactive Communication in the Presence of Adversarial Noise. *SIAM J. Comput.* 46, 1 (2017), 388–428. <https://doi.org/10.1137/141002001>
- [8] Mark Braverman, Klim Efremenko, Ran Gelles, and Bernhard Haeupler. 2017. Constant-Rate Coding for Multiparty Interactive Communication Is Impossible. *J. ACM* 65, 1, Article 4 (Dec. 2017), 41 pages. <https://doi.org/10.1145/3050218>
- [9] M. Braverman, R. Gelles, J. Mao, and R. Ostrovsky. 2017. Coding for Interactive Communication Correcting Insertions and Deletions. *IEEE Transactions on Information Theory* 63, 10 (Oct 2017), 6256–6270. <https://doi.org/10.1109/TIT.2017.2734881>
- [10] M. Braverman and A. Rao. 2014. Toward Coding for Maximum Errors in Interactive Communication. *Information Theory, IEEE Transactions on* 60, 11 (Nov 2014), 7248–7255. <https://doi.org/10.1109/TIT.2014.2353994>
- [11] Keren Censor-Hillel, Ran Gelles, and Bernhard Haeupler. 2018. Making Asynchronous Distributed Computations Robust to Channel Noise. In *9th Innovations in Theoretical Computer Science Conference (ITCS 2018) (Leibniz International Proceedings in Informatics (LIPIcs))*, Anna R. Karlin (Ed.), Vol. 94. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, Dagstuhl, Germany, 50:1–50:20. <https://doi.org/10.4230/LIPIcs.ITCS.2018.50>
- [12] Klim Efremenko, Elad Haramaty, and Yael Kalai. 2018. Interactive Coding with Constant Round and Communication Blowup. *Electronic Colloquium on Computational Complexity (ECCC)* 25 (2018), 54. <https://eccc.weizmann.ac.il/report/2018/054>
- [13] Ran Gelles. 2017. Coding for Interactive Communication: A Survey. *Foundations and Trends® in Theoretical Computer Science* 13, 1–2 (2017), 1–157. <https://doi.org/10.1561/04000000079>
- [14] R. Gelles, B. Haeupler, G. Kol, N. Ron-Zewi, and A. Wigderson. 2018. Explicit Capacity Approaching Coding for Interactive Communication. *IEEE Transactions on Information Theory* 64, 10 (Oct 2018), 6546–6560. <https://doi.org/10.1109/TIT.2018.2829764>
- [15] R. Gelles and Y. T. Kalai. 2019. Constant-Rate Interactive Coding Is Impossible, Even in Constant-Degree Networks. *IEEE Transactions on Information Theory* 65, 6 (June 2019), 3812–3829. <https://doi.org/10.1109/TIT.2019.2904576>
- [16] Ran Gelles, Yael Tauman Kalai, and Govind Ramnarayan. 2019. Efficient Multiparty Interactive Coding for Insertions, Deletions and Substitutions. *CoRR preprint arXiv:1901.09863* (2019). <http://arxiv.org/abs/1901.09863>
- [17] Ran Gelles, Ankur Moitra, and Amit Sahai. 2011. Efficient and Explicit Coding for Interactive Communication. In *Proceeding of the IEEE Symposium on Foundations of Computer Science (FOCS '11)*, 768–777. <https://doi.org/10.1109/FOCS.2011.51>
- [18] Ran Gelles, Ankur Moitra, and Amit Sahai. 2014. Efficient Coding for Interactive Communication. *Information Theory, IEEE Transactions on* 60, 3 (March 2014), 1899–1913. <https://doi.org/10.1109/TIT.2013.2294186>
- [19] Mohsen Ghaffari and Bernhard Haeupler. 2014. Optimal Error Rates for Interactive Coding II: Efficiency and List Decoding. In *Proceedings of the IEEE Symposium on Foundations of Computer Science (FOCS '14)*, 394–403. <https://doi.org/10.1109/FOCS.2014.49>
- [20] Bernhard Haeupler. 2014. Interactive Channel Capacity Revisited. In *Proceedings of the IEEE Symposium on Foundations of Computer Science (FOCS '14)*, 226–235. <https://doi.org/10.1109/FOCS.2014.32>
- [21] Bernhard Haeupler, Amirbehshad Shahrashbi, and Ellen Vitercik. 2017. Synchronization Strings: Channel Simulations and Interactive Coding for Insertions and Deletions. *CoRR preprint arXiv:1707.04233* (2017). <http://arxiv.org/abs/1707.04233>
- [22] William M. Hoza. 2015. Private communication.
- [23] William M. Hoza and Leonard J. Schulman. 2016. The Adversarial Noise Threshold for Distributed Protocols. In *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms*, 240–258. <https://doi.org/10.1137/1.9781611974331.ch18>
- [24] Abhishek Jain, Yael Tauman Kalai, and Allison Lewko. 2015. Interactive Coding for Multiparty Protocols. In *Proceedings of the 6th Conference on Innovations in Theoretical Computer Science, ITCS '15*, 1–10. <https://doi.org/10.1145/2688073.2688109>
- [25] Gillat Kol and Ran Raz. 2013. Interactive channel capacity. In *STOC '13: Proceedings of the 45th annual ACM Symposium on theory of computing*, 715–724. <https://doi.org/10.1145/2488608.2488699>
- [26] Joseph Naor and Moni Naor. 1993. Small-Bias Probability Spaces: Efficient Constructions and Applications. *SIAM J. Comput.* 22 (1993), 838–856.
- [27] Sridhar Rajagopalan and Leonard Schulman. 1994. A coding theorem for distributed computation. In *STOC '94: Proceedings of the twenty-sixth annual ACM symposium on Theory of computing*, 790–799. <https://doi.org/10.1145/195058.195462>
- [28] Leonard J. Schulman. 1992. Communication on noisy channels: a coding theorem for computation. *Foundations of Computer Science, Annual IEEE Symposium on* (1992), 724–733. <https://doi.org/10.1109/SFCS.1992.267778>
- [29] Leonard J. Schulman. 1996. Coding for interactive communication. *IEEE Transactions on Information Theory* 42, 6 (1996), 1745–1756. <https://doi.org/10.1109/18.556671>
- [30] Alexander A. Sherstov and Pei Wu. 2017. Optimal Interactive Coding for Insertions, Deletions, and Substitutions. In *2017 IEEE 58th Annual Symposium on Foundations of Computer Science (FOCS)*, 240–251. <https://doi.org/10.1109/FOCS.2017.30>