Algorithms for Noisy Broadcast with Erasures

Ofer Grossman
EECS Departent, MIT, Cambridge, MA, USA
ofer.grossman@gmail.com

Bernhard Haeupler
Computer Science Department, Carnegie Mellon University, Pittsburgh, PA, USA
haeupler@cs.cmu.edu

Sidhanth Mohanty
Computer Science Department, Carnegie Mellon University, Pittsburgh, PA, USA
sidhanth@cmu.edu

Abstract

The noisy broadcast model was first studied by [10] where an n-character input is distributed among n processors, so that each processor receives one input bit. Computation proceeds in rounds, where in each round each processor broadcasts a single character, and each reception is corrupted independently at random with some probability p. [10] gave an algorithm for all processors to learn the input in $O(\log \log n)$ rounds with high probability. Later, a matching lower bound of $\Omega(\log \log n)$ was given by [11].

We study a relaxed version of this model where each reception is erased and replaced with a '?' independently with probability p, so the processors have knowledge of whether a bit has been corrupted. In this relaxed model, we break past the lower bound of [11] and obtain an $O(\log^* n)$-round algorithm for all processors to learn the input with high probability. We also show an $O(1)$-round algorithm for the same problem when the alphabet size is $\Omega(\text{poly}(n))$.

2012 ACM Subject Classification Theory of computation → Distributed computing models

Keywords and phrases noisy broadcast, error correction, erasures, distributed computing with noise

Digital Object Identifier 10.4230/LIPIcs.ICALP.2018.153

1 Introduction

In recent years, it is becoming increasingly common for computational tasks to be performed by multiple processors in a distributed fashion. The communication channels of these networks may have imperfections, which introduces noise to the system.

A formal version of a noise model was proposed by [7]: There are n processors: $1, 2, \ldots, n$ and each processor is given a bit. In each round, every processor broadcasts a bit to all other processors. Every processor will receive the correct message with some probability, and may receive a different (corrupted) message independently with probability $p < 1/2$ (i.e., each reception gets corrupted with probability p). The goal is for the processors to collectively compute the XOR of all their inputs. An algorithm that takes $O(\log \log n)$ rounds for all processors to learn the full input (and hence the XOR as well) was found by [10]. A matching lower bound of $\Omega(\log \log n)$ rounds was proven by [11].

All of the prior works were concerned with *substitution errors*. In this paper, we study such networks in the presence of *erasure errors*, where instead of messages getting corrupted into other messages, instead messages may get dropped. Specifically, we study the following model: in a single round each processor can broadcast a single bit b to all other processors. For each ordered pair (i, j), independently with some probability p, the character that i


© Ofer Grossman, Bernhard Haeupler, and Sidhanth Mohanty;
licensed under Creative Commons License CC-BY
45th International Colloquium on Automata, Languages, and Programming (ICALP 2018).
Editors: Ioannis Chatzigiannakis, Christos Kaklamanis, Dániel Marx, and Donald Sannella;
Article No. 153; pp. 153:1–153:12


Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

transmitted is not received by j and a '?' is received instead. In other words, there is a string $X \in \{0,1\}^n$ and processor i is given the ith bit of X, called x_i, and the goal is for each processor to learn X using as few rounds of communication as possible. We call our noise model the *erasure model*.

1.1 Our results

We show that for any alphabet, each processor can learn the inputs of all other processors with high probability within $O(\log^* n)$ rounds. At the high level, the algorithm involves recursively running the protocol on groups of size $\log n$, and having each group encode its input using a constant rate and constant relative distance error correcting code. Then, the group collectively transmits this encoded string within a constant number of rounds. It can be shown that with high probability every processor receives enough bits to decode the group's input. There are groups for which not enough processors learn the full string (i.e., the recursive call fails), and some technical steps are needed to handle these 'failed groups'. The protocol is described in full detail in section 2.

We note that in the presence of *substitution errors*, it was proven in [11] that $\Omega(\log \log n)$ rounds are required for all processors to learn the whole input. Since we show a $O(\log^*(n))$ algorithm for the problem in the presence of *erasure errors*, this shows a fundamental difference between substitution errors and erasure errros in the broadcast model.

We then show that when the alphabet is of polynomial size, there is an $O(1)$ round algorithm for every processor to learn the full input. The algorithm involves treating the alphabets as elements of a finite field $\mathbb{F}_q$, and simulating multiplying the input vector with an appropriate random matrix. Then , the processors receive a random system of linear equations which one can show has a unique solution with high probability.

We then show that any symmetric function of the input can be computed within a constant number of rounds via computing the Hamming weight.

1.2 Related Work

A related problem was studied in [10] where the broadcast model assumed was sequential, where in one round only one processor can broadcast a bit. Additionally, the noise model assumed was that of bit flips instead of erasures. That is, each transmitted bit is independently flipped with probability p on the receiving end. In their model, [10] shows that all the processors can learn the entire input within $O(\log \log n)$ rounds. However, it left open the question of whether a faster protocol was possible.

The model of [10] was studied further in [11] where a lower bound of $\Omega(n \log \log n)$ was proven for the total number of broadcasts, thereby establishing that Gallager's protocol is optimal up to constant factors. The lower bound is proved via a reduction to another model called the generalized noisy decision tree, which is a variant of the noisy decision tree model introduced in [9]. [11] also studies whether more efficient protocols exist when the processors only want to compute some specific function on the entire input and shows that the Hamming weight can be computed with constant probability within $O(n)$ broadcasts.

We note that it follows from the lower bound in [11] that in a variant of our model where one considers substitution errors instead of erasure errors, any protocol from which all the processors learn the entire input must take $\Omega(\log \log n)$ rounds. In light of this lower bound, our result of an $O(\log^* n)$ protocol is interesting, as it shows a fundamental difference between substitution and erasure errors in this broadcast model.

Recently, a work by Efremenko, Kol, and Saxena [6] showed that under a model where the processors can adaptively choose which processor will speak in each round, the lower bound of [11] breaks down.

Note that the work of Gallager [10] shows that in the substitution model where a single processor broadcasts to the rest in a round, any function can be computed within $O(n \log \log n)$ rounds. A work by Kushilevitz and Mansour [14] studies the question of which Boolean functions can be computed within $O(n \log \log n)$ broadcasts. They determine that threshold functions can be computed with constant probability within $O(n)$ broacasts.

A paper by Feige and Killian [8] studied a harsher noise model than [10], where an adversary can arbitrarily 'uncorrupt' arbitrary corrupted bits, causing the noise to lose structure. In this harsher model, they show an $O(\log^* n)$ round protocol to compute the OR of all input bits. Newman [16] studies another noise model where each bit transmitted is independently flipped with an unknown probability that is at most p and gives algorithms that use $O(n)$ broadcasts and $O(\log^* n)$ rounds for certain classes of Boolean functions, including OR, AND, and functions with linear size AC_0 formulas.

In [1], the authors show efficient protocols to handle errors in the UCAST model, in which instead of broadcasting bits, a processor can send a different message to each other processor. They also show efficient protocols to handle errors when the communication network has certain expansion properties. For general graphs of low degree, a protocol for handling errors was found in [17], which was later shown to be optimal in [3].

Our model in the absence of errors is known as the Broadcast Congested Clique, which is a computational model often studied in distributed computing (see for example, [5, 15, 4, 2, 12]. In this model, n processors each get a piece of the input, and they work together to compute some function of this shared input. Computation proceeds in rounds, where in each round each processor can broadcast a short message to all other processors. Our work can be interpreted as showing that when using messages of constant size, every protocol in the Broadcast Congested Clique can be made resilient to erasure errors with a blowup of only $O(\log^*(n))$. In the case where messages are of logarithmic size, we show the Broadcast Congested Clique can be made resilient to erasure errors with only a constant blowup.

1.3 Notation and conventions

In this section, we state some notational conventions we use. First, we describe the computational model (without erasures), and then we formally define the model we consider with erasures.

The Computational Model: In a setting with n processors, each processor is identified with a distinct number in $[n]$. Given a string X, which we denote using an upper case character, we write the ith bit as x_i, using the corresponding lower case character. To denote the substring of X starting at position i and ending in position j we write $X_{[i,j]}$. When we wish to compute some function of a n-bit string X using n processors, assume x_i is provided as input to processor i. In the description of algorithms, $\textsc{Algo}(x_1, \ldots, x_n)$ refers to an algorithm that runs on n processors where the ith processor is given x_i as input.

In all our algorithms, we assume that each broadcast is repeated γ times where γ is some appropriately chosen constant.

Formally, we have:

▶ **Definition 1.** We let the *noisy parallel broadcast model* be a model of computation where there are n processors $P_1, \ldots, P_n$, and P_i receives input bit x_i. In each round of computation, each processor can broadcast one bit to all other processors. Each reception is corrupted

with some constant probability $0 \leq p < 1$, in which case the character '?' is received instead of the bit which was sent.

In this paper, we study the complexity of computing certain functions in the above model. Specifically, for constant erasure probability p we show a bound of $O(\log^*(n))$ for computing any function, and a bound of $O(1)$ for symmetric functions.

As part of our algorithm we use error correcting codes, so we include standard results and notations for codes below: **Error Correcting Codes:** An error correcting code is described by functions $\mathsf{Enc} : \{0,1\}^n \to \{0,1\}^k$ and $\mathsf{Dec} : \{0,1\}^k \to \{0,1\}^n$ with the property that $\mathsf{Dec}(x)$ maps to the string y for which $\mathsf{Enc}(y)$ is the closest string to x in $\mathcal{C} = \text{Image}(\mathsf{Enc})$.

The *rate* of an error correcting code is defined as $\frac{k}{n}$ and the *relative distance* is defined as $\frac{\min_{x,y \in \mathcal{C}} d(x,y)}{k}$. The quantity $\frac{d(x,y)}{2}$ is referred to as the *decoding radius.*

We use the result of [13] that error correcting code families of constant rate and constant relative distance exist. In particular, for the sake of this paper, we assume the existence of an error correcting code family E with relative distance 0.25 and rate some absolute constant K.

2 An $O(\log^* n)$ algorithm for computing any function

We consider the following message-passing model. There are n processors, and in each round, every processor transmits a single bit b to all other processors. Each processor receives each bit independently and at random with probability $1 - p$. With probability p, the character '?' is received instead. If each processor starts with a single input bit, we ask how many rounds are required so that every processor knows all input bits with high probability. We show a bound of $O(\log^*(n))$ for this problem. Specifically, we will show:

▶ **Theorem 2.** *For every $0 \leq p < 1$, there is an algorithm in the noisy broadcast parallel erasure model that computes ID_n with high probability within $O(\log^*(n) \log \frac{1}{1-p})$ rounds.*

Without loss of generality, we assume that $p \leq 0.01$, since for any erasure probability $p < 1$, repeating each message $O(\log \frac{1}{1-p})$ times can be used to effectively lower the probability of receiving '?'. We describe our algorithm for the case where the alphabet $\Sigma = \{0,1\}$. The protocol generalizes to larger alphabets in a straightforward manner.

We describe a protocol for n processors with the guarantees: at the end of the protocol, all n processors can output the full string C with probability at least $1 - \frac{1}{n^5}$, and if the protocol fails (that is, there is some processor who cannot output the full string C), then all n processors can output '$\perp$' with probability at least $1 - \frac{1}{2^{7n}}$. For the rest of this section, we assume $n \geq n_0$ for a sufficiently large n_0.

We begin by describing algorithms for simpler subproblems.

▶ **Lemma 3.** *Let b_i be the input to processor i, and let the erasure probability p be .01. Then there is an $O(1)$-round algorithm and an absolute constant α such that all processors output the AND of all b_i with probability at least $1 - 2^{-\alpha n}$.*

Proof. **Algorithm:** The algorithm is as follows: in each round, a processor i broadcasts '0' either if $x_i = 0$ or if processor i has received at least one '0' in at least one of the previous rounds. Otherwise, processor i boradcasts 1. This is repeated for 100 rounds.

Processor j's output is the AND of all bits it received.

Analysis: First, note that if all of the $b_i = 1$, then all processors must output 1, no matter what messages were corrupted, since all received bits of all processors must be 1.

Algorithm 1: EQUALITYTEST$(S_1, \ldots, S_n)$.

Enc is the encoding function of a code $\mathcal{C}$ with relative distance 0.25 and constant rate K.

1. Transmit $(\mathsf{Enc}(S_i))_{[(i-1)K+1,iK]}$ over K rounds
2. Let $A_{t,i}$ be the K-bit string received from processor t and $A_i = A_{1,i}A_{2,i}\ldots A_{n,i}$. Set c_i to 1 if Hamming distance between A_i and $\mathsf{Enc}(S_i)$ is at most $0.06Kn$ and 0 otherwise
3. The processors run the AND protocol from Lemma 3 and output the AND of all c_i

Now, suppose there is an i for which $b_i = 0$. Let t be the number of processors that received the transmission of i in the first round. The probability that processor j receives only 1s in the second round is at most p^t.

We can use Hoeffding's inequality to obtain

$$\mathbf{Pr}\left[t < \frac{n}{2}\right] \leq e^{-\alpha' n}$$

for some constant α'. Thus, the probability that there is some j that received only 1's even if there is a processor with a 0 is at most $n(e^{-\alpha' n} + p^{n/2})$, bounded above by $e^{-\alpha n}$ for a constant α. ◀

We note that the above protocol does not work in the substitution model (the model where a message may be flipped with small probability, as opposed to being corrupted to a '?'). In fact, in [11] it was proven that computing the AND function with high probability in the substitution model requires $\Omega(\log \log n)$ rounds.

We next show an $O(1)$ round algorithm for EQUALITY TESTING. Each processor is given an n-bit string S_i as input, and the goal is for all processors to output 1 if all their inputs are equal and 0 otherwise with probability at least $1 - 2^{-\Omega(n)}$. Unless otherwise specified, each step of the algorithm is from the view of processor i. Roughly speaking, this step will be used in the main algorithm to verify that all processors end up with the same output string S.

▶ **Lemma 4.** *When the erasure probability $p \leq .01$, Algorithm 1 correctly solves* EQUALITY TESTING *with probability at least $1 - 2^{-\beta n}$ for some absolute constant β.*

Proof. Let A be the string collectively transmitted by all processors in Step 1. We know

$$\begin{aligned} d(\mathsf{Enc}(S_i), \mathsf{Enc}(S_j)) &\leq d(\mathsf{Enc}(S_i), A_i) + d(A_i, A) + d(A, A_j) + d(A_j, \mathsf{Enc}(S_j)) \\ &\leq 2(d(\mathsf{Enc}(S_i), A_i) + d(\mathsf{Enc}(S_j), A_j)) \end{aligned}$$

where the second inequality is because $d(A, A_i)$ is the number of '?'s received, and lower bounds $d(\mathsf{Enc}(S_i), A_i)$.

If both $d(\mathsf{Enc}(S_i), A_i)$ and $d(\mathsf{Enc}(S_j), A_j)$ are at most $0.06Kn$, then $d(\mathsf{Enc}(S_i), \mathsf{Enc}(S_j))$ is at most $0.24Kn$, but since they are codewords of a code with relative distance 0.25, $\mathsf{Enc}(S_i) = \mathsf{Enc}(S_j)$, implying $S_i = S_j$. So if there is a pair i, j with $S_i \neq S_j$, then either c_i or c_j must be 0. And then from Lemma 3, with probability at least $1 - e^{-\alpha n}$, the processors correctly detect that there is a c_i equal to 0.

On the other hand, if all the strings are indeed equal, then c_j is 0 only if processor j receives fewer than $0.94Kn$ bits. We upper bound the probability that this happens by using Chernoff bound along with a union bound over all processors.

$$n\mathbf{Pr}[\text{processor } i \text{ receives fewer than } 0.88Kn \text{ bits}] \leq ne^{-\alpha'' n} \leq e^{-\alpha' n}$$

where α' is some constant. We let $\beta = \min\{\alpha, \alpha'\}$. ◀

Let $X = x_1x_2 \dots x_n$ be the input string and processor i is given x_i and is required to output a tuple (X_i, s_i), where X_i an n-bit string and s either 1, indicating success or 0, indicating failure, with the goal of having all $X_i = X$ and all $s_i = 1$. We say that an algorithm on a group of processors **succeeded** if $X_i = X$ and $s_i = 1$ for all i, **failed with knowledge** if $r_i = 0$ for all i, and **failed without knowledge** otherwise. We describe an algorithm for this problem where each step is from the view of processor i unless otherwise specified. Recall that each broadcast is repeated γ times to effectively reduce the erasure probability p to be at most .01. For simplicity, we assume that n is a power of 2, and so $\log n$ is an integer. It is easy to generalize the algorithm to all values of n.

At the high level, the algorithm proceeds as follows. We partition the processors into $n/\log n$ sets of size $\log n$ each (Step 2a). Then, we recursively compute the input on each of these subsets. Now, some of these subsets will have succeeded, and some will have failed. For the ones that failed, we now recompute the input, but this time we add more processors to be "helper processors". That is, the processors which succeeded in the recursive calls will now be used to aid the processors who failed in the recursive call by sending messages on their behalf. This can be seen in Step 2g, where the processor sends x_{ℓ_i}, which is the input to a processor which failed on the recursive call. This idea of using successful processors to help others who failed helps ensure that within a constant number of tries, with high probability all input bits will be known.

We now prove the following proposition, from which Theorem 2 immediately follows.

▶ **Proposition 5.** *Algorithm 2 runs in $O(\log^* n)$ rounds, succeeds (i.e., each processor outputs $(X, 1)$, where X is the input to all processors) with probability at least $1 - \frac{1}{n^5}$ and fails without knowledge with probability at most $\frac{1}{2^{7n}}$.*

Proof. We list conditions under which the protocol definitely succeeds, and show all these conditions hold with probability at least $1 - \frac{1}{n^5}$. Define R as $r'_1r'_2 \dots r'_n$ from the output of Step 2a. Define M_s as all j such that $\frac{n(s-1)}{z} < j \leq \frac{ns}{z}$ where z is the number of 0's in R.

The protocol definitely succeeds if the following conditions hold:

1. All but at most $\frac{n}{\log^3 n}$ groups succeed in the recursive call of Step 2a.
2. No group fails without knowledge in the recursive call of Step 2a, and $R_i = R$ for all i.
3. For all j such that $(R)_j = 0$, for all i, processor i receives at least one transmission from a processor in $M_{\ell_j,i}$ in Step 2g where the ℓ_jth 0 in R occurs at $(R)_j$.
4. Each processor receives at least $0.88K \log n$ bits from each successful group in at least one transmission in Step 2d of the algorithm.

Indeed, for any j in a successful group, all processors correctly learn the input to processor j because Condition 4 is met. By Condition 2, for fixed s, $M_{s,i}$ is the same for all i since $M_{s,i}$ depends on R_i. For any j in a failed group, by Condition 2, $(R)_j = (R_i)_j = 0$, and by Condition 3, each processor receives at least one transmission of processor j's input in Step 2g and so all processors correctly learn the input to processor j.

We now proceed with showing a lower bound on the probability that all of these conditions hold.

We can see that Condition 1 holds with probability at least $1 - \frac{1}{n^6}$ since by Chernoff bounds, the number of failed groups exceeds $\frac{n}{\log^3 n}$ with probability at most $\frac{1}{n^6}$.

Now suppose Condition 1 holds. A group fails without knowledge with probability at most $\frac{1}{n^7}$ by the guarantees of the protocol. The probability that there exists a group that failed without knowledge, by the union bound, is therefore at most $\frac{1}{n^6}$. If no group failed without knowledge, the only way R_i cannot equal R is if there is a group $M_{j,i}$ that processor i did not receive a single bit from. The probability that processor i does not receive a single

Algorithm 2: LearnInput$(x_1, \ldots, x_n)$.

Enc is the encoding function of a code $\mathcal{C}$ with relative distance 0.25 and constant rate K

1. **Base Case:** If $n < 100$
   a. Transmit x_i repeatedly 100 times, and set string S_i as per

$$(S_i)_j = \begin{cases} b & \text{if } b \text{ was received in any transmission from } j \\ \text{random bit} & \text{if all transmissions from } j \text{ are `?'} \end{cases}$$

   and go to Step 3a.
2. **Recursive Step:**
   a. Recursively obtain $(X'_i, r'_i) = \text{LearnInput}\left(x_{\lfloor \frac{i}{\log n} \rfloor \log n+1}, \ldots, x_{\lfloor \frac{i}{\log n} \rfloor \log n+\log n}\right)$. We call this set of processors the *group* of i.
   b. Broadcast r'_i
   c. Set R_i by setting $(R_i)_j$ to 1 if only 1's were received from j's group (i.e., from the processors which j computed the recursive call with) and 0 otherwise, for each $j \in [n]$.
   d. Let $i' = i \bmod \log n$ and transmit $\mathsf{Enc}(X'_i)_{[(i'-1)K+1, i'K]}$ over the next K rounds.
   e. Let z_i be the number of zeros in R_i and let $j = \left\lceil \frac{iz_i}{n} \right\rceil$ and let ℓ_i be the index of the jth zero in R_i. Create set $M_{s,i}$ to be all t such that

$$\frac{n(s-1)}{z_i} < t \leq \frac{ns}{z_i}$$

   f. Transmit x_i.
   g. Broadcast what was received from ℓ_i, which is either '?' or x_{ℓ_i}. Let $M'_{j,i}$ be the set of characters received from $M_{j,i}$.
   h. Set X_i by setting $(X_i)_j$ to x_i if $j = i$, by decoding the bits received in Step 2d if $(R_i)_j = 1$ and at least $0.88K \log n$ bits were received from the group of j, to a random bit if $(R_i)_j = 1$ and fewer than $0.88K \log n$ bits were received from group j in Step 2d, and to $\mathbf{1}_{1 \in M'_{j,i}}$ if $(R_i)_j = 0$. Proceed to Step 3a.
3. **Verification of output**
   a. Obtain $v_i = \text{EqualityTest}(X_1, \ldots, X_n)$ and output (X_i, v_i).

bit from this group is $p^{\gamma \log n}$, which for appropriate γ is at most $\frac{1}{n^8}$. Thus, the probability that there is some i, j pair such that processor i does not receive a single bit from group j is at most $\frac{1}{n^6}$ by a union bound. So the probability that Condition 2 is not met (given that Condition 1 is met) is at most $\frac{2}{n^6}$.

Note that $R_i = R$ means $M_s = M_{s,i}$ for all i. It follows from Chernoff bounds that the number of processors in M_s that receive the bit transmitted by processor s is at least $\log n$ with probability at least $1 - \frac{1}{n^6}$. The probability that processor i does not receive any bits from processors in M_s in any of the repetitions of Step 2g is at most $p^{\gamma \log n}$, which can be made smaller than $\frac{1}{n^8}$ by setting γ to be large enough. Now by taking a union bound over all pairs (i, s) we can conclude that Condition 3 does not hold with probability at most $\frac{1}{n^6}$.

The probability that processor i receives fewer than $0.88K \log n$ bits from group j in all repetitions of Step 2d is at most $\frac{1}{n^{c\gamma}}$ for some constant c by Chernoff bounds. A union bound across all processor-group pairs tells us that Condition 4 does not hold with probability at most $\frac{1}{n^{c\gamma-2}}$ which can be made smaller than $\frac{1}{n^6}$ with large enough γ.

Algorithm 3: LearnInputLargeAlphabet$(x_1, \ldots, x_n)$.

Let F be a function that encodes subsets of $[6 \log n]$ as elements of $\mathbb{F}_q$

1. Let $k = \lfloor 6 \log n \rfloor$ and determine $B_i = \{\frac{nj}{k} + 1, \ldots, \frac{n(j+1)}{k}\}$, where j is chosen such that $i \in B_i$
2. Broadcast x_i for 10 rounds
3. For each t from 1 to 10 and for each processor in B_i from which an entry was received in round t of Step 2, choose the processor with probability $\frac{1}{2(1-p)}$ and choose i with probability $\frac{1}{2}$. Let $T_{t,i}$ be the set of chosen elements.
4. For the next 20 rounds, processor i transmits all the $\sum_{b \in T_{t,i}} x_b$ (where the x_b are added as elements of $\mathbb{F}_q$) and $F(T_{t,i})$
5. Output X_i consistent with all received pairs $\left(\sum_{b \in T_{t,i}} x_b, F(T_{t,i})\right)$. If there is more than one possibility for such an X_i, pick one at random.

Based on the bounds we obtained on the probability that each of Conditions 1, 2, 3, 4 don't hold, we can conclude that the probability that all the conditions hold is at least $1 - \frac{1}{n^5}$.

It remains to show that the probability that the processors failed without knowledge is at most 2^{-7n}. If there is X_i such that $X_i \neq X$, then it differs from X in some index j, which means $(X_i)_j \neq (X_j)_j$ by construction of X_j implying $X_i \neq X_j$. Thus, a failure without knowledge happens only if Step 3a fails, which happens with probabilty at most $e^{-\beta n}$, which can be made smaller than 2^{-7n} by choosing the number of repetitions γ to be a large enough constant.

The number of rounds this algorithm takes is given by $T(n)$, which satisfies the recurrence relation $T(n) = T(\log n) + L$ where L is a constant and with base case $T(100) = O(1)$, which solves to $T(n) = O(\log^* n)$. ◀

3 An $O(1)$ algorithm for large alphabets

For large alphabets, in the regime where the alphabet Σ is $\mathbb{F}_q$ and $q = \text{poly}(n)$, we give a constant round algorithm to have all processors learn the input X with probability at least $1 - \frac{1}{\text{poly}(n)}$. Unless otherwise specified, the algorithm is from the view of processor i. While our algorithm works for any q that is polynomial in n, for simplicity of exposition we assume $q \geq n^6$ and that q is a prime.

▶ **Theorem 6.** *With probability at least $1 - \frac{1}{\text{poly}(n)}$, after running Algorithm 3, all processors will know all other processors' inputs. Furthermore, the algorithm terminates within $O(1)$ rounds.*

As a first ingredient towards proving Theorem 6, we prove the following lemma.

▶ **Lemma 7.** *If A is a $5k \times k$ random binary matrix where each entry is i.i.d. generated by flipping a fair coin, then with probability at least $1 - e^{-0.4k}$, A is full rank.*

Proof. Suppose V is a subspace of $\mathbb{F}_q^k$ that is not equal to all of $\mathbb{F}_q^k$, then we can find standard basis vector e_i that is not in V. Then for any binary vector v, consider v' with the bit at the i-th coordinate flipped. Either v or v' is not in V, which means at least half of the binary vectors are not in V, which means each new vector has probability at least $\frac{1}{2}$ of not being in

V. If we let $V = \mathsf{span}\{\text{vectors drawn so far}\}$, then each draw has a probability at least $\frac{1}{2}$ of increasing the dimension. Suppose we flip $5k$ coins, the probability that the number of heads is at most k is an upper bound on the probability of the span of $5k$ randomly drawn vectors not being the whole space.

By Chernoff bounds, this probability is at most $e^{-0.4k}$. ◀

Proof of Theorem 6. Each $T_{t,i}$ is a uniformly random subset of input bits of set B_i. Let x_{B_i} be a k-dimensional vector of the inputs to processors in B_i, then the transmitted characters in Round 5 are of the form $(\langle a_{B_i}, x_{B_i}\rangle, F(T_{t,i}))$ where a_{B_i} is a random binary vector, and $F(T_{t,i})$ is an encoding of a_{B_i}. The transmitted characters can be viewed as elements in the vector Ax_{B_i}, where A is a matrix whose rows are the a_{B_i}. A single processor's output of x_{B_i} is given by sampling rows of the equation $Ay_{B_i} = Ax_{B_i}$ where y_{B_i} is indeterminate and solving for y_{B_i}. If the number of sampled rows is at least $5k$, then from Lemma 7 the probability that the sampled rows span $\mathbb{F}_q^k$ and hence give a unique solution to y_{B_i} is at least $1 - \frac{1}{n^{2.4}}$.

The probability that the number of sampled rows for a group is less than $5k$ can be upper bounded by $\frac{1}{n^5}$ using Chernoff bounds.

So by union bound over all group-processor pairs (i.e., all pairs (i, B_j)), we get a $\frac{1}{\mathrm{poly}(n)}$ upper bound on the failure probability. ◀

References

1. Noga Alon, Mark Braverman, Klim Efremenko, Ran Gelles, and Bernhard Haeupler. Reliable communication over highly connected noisy networks. In *Proceedings of the 2016 ACM Symposium on Principles of Distributed Computing*, pages 165–173. ACM, 2016.
2. Florent Becker, Antonio Fernandez Anta, Ivan Rapaport, and Eric Reémila. Brief announcement: A hierarchy of congested clique models, from broadcast to unicast. In *Proceedings of the 2015 ACM Symposium on Principles of Distributed Computing*, pages 167–169. ACM, 2015.
3. Mark Braverman, Klim Efremenko, Ran Gelles, and Bernhard Haeupler. Constant-rate coding for multiparty interactive communication is impossible. *Journal of the ACM (JACM)*, 65(1):4, 2017.
4. Keren Censor-Hillel, Petteri Kaski, Janne H Korhonen, Christoph Lenzen, Ami Paz, and Jukka Suomela. Algebraic methods in the congested clique. In *Proceedings of the 2015 ACM Symposium on Principles of Distributed Computing*, pages 143–152. ACM, 2015.
5. Andrew Drucker, Fabian Kuhn, and Rotem Oshman. On the power of the congested clique model. In *Proceedings of the 2014 ACM symposium on Principles of distributed computing*, pages 367–376. ACM, 2014.
6. Klim Efremenko, Gillat Kol, and Raghuvansh Saxena. Interactive coding over the noisy broadcast channel. In *Electronic Colloquium on Computational Complexity (ECCC)*, volume 24, page 93, 2017.
7. A El Gamal. Open problems presented at the 1984 workshop on specific problems in communication and computation sponsored by bell communication research. *Open Problems in Communication and Computation*, 1987.
8. Uriel Feige and Joe Kilian. Finding or in a noisy broadcast network. *Information Processing Letters*, 73(1-2):69–75, 2000.
9. Uriel Feige, Prabhakar Raghavan, David Peleg, and Eli Upfal. Computing with noisy information. *SIAM Journal on Computing*, 23(5):1001–1018, 1994.
10. Robert G Gallager. Finding parity in a simple broadcast network. *IEEE Transactions on Information Theory*, 34(2):176–180, 1988.

11 Navin Goyal, Guy Kindler, and Michael Saks. Lower bounds for the noisy broadcast problem. *SIAM Journal on Computing*, 37(6):1806–1841, 2008.

12 Tomasz Jurdzinski and Krzysztof Nowicki. Msf and connectivity in limited variants of the congested clique. *arXiv preprint arXiv:1703.02743*, 2017.

13 Jørn Justesen. Class of constructive asymptotically good algebraic codes. *IEEE Transactions on Information Theory*, 18(5):652–656, 1972.

14 Eyal Kushilevitz and Yishay Mansour. Computation in noisy radio networks. In *SODA*, volume 98, pages 236–243, 1998.

15 Pedro Montealegre and Ioan Todinca. Deterministic graph connectivity in the broadcast congested clique. *arXiv preprint arXiv:1602.04095*, 2016.

16 Ilan Newman. Computing in fault tolerance broadcast networks. In *Computational Complexity, 2004. Proceedings. 19th IEEE Annual Conference on*, pages 113–122. IEEE, 2004.

17 Sridhar Rajagopalan and Leonard Schulman. A coding theorem for distributed computation. In *Proceedings of the twenty-sixth annual ACM symposium on Theory of computing*, pages 790–799. ACM, 1994.

A An $O(1)$ protocol for computing any symmetric function

We show that any symmetric function can be computed within $O(1)$ rounds in the model. Symmetric functions are functions whose value doesn't change under permutation of the input bits. In other words, these functions only depend on the Hamming weight of the input string. Hence, an algorithm for every processor to learn the Hamming weight of the string leads to an algorithm to compute any symmetric function. Our algorithm is inspired by a similar algorithm (for a different model) of [11].

▶ **Theorem 8.** *There is an $O(1)$ round algorithm in the noisy broadcast parallel erasure model that computes* Hamming Weight(X) *with probability at least* 0.75*.*

Our algorithm proceeds in two phases:

1. Divide the interval $[0, n]$ into subintervals of length $c\sqrt{n}$ and find which interval the Hamming weight belongs to.
2. Figure out exactly which integer in the interval is the Hamming weight.

More precisely, the first step will give us three intervals, and we will show for at least two of these intervals, with high probability all processors will end up with the same interval. Then, we will run the second step (where we pinpoint the exact Hamming weight) on each of the three intervals, and take a majority vote to compute the final output.

We describe the first step below:

▶ **Lemma 9.** *With probability at least $1 - \exp(-\Omega(n))$, for at least two t in $\{1, 2, 3\}$, all $C_{i,t}$ outputted in Step 4 of Algorithm 4 are equal and correspond to an interval containing* Hamming Weight(X)*.*

Proof. By Chernoff bounds, the probability that h_i deviates from the truth by $t\sqrt{n}$ is at most e^{-Ct^2} for an absolute constant C. This can be made smaller than 0.01 with appropriate choice of a constant t. Then for at least two values of s, h_i lies in the correct interval in $\mathcal{B}_s$ with probability at least 0.99. Without loss of generality, say this happens for $s = 0$ and $s = 1$. Using Chernoff bounds, we can show that for some constant c, with probability at least $1 - \exp(-\Omega(n))$, at least 0.95 fraction of the processors decode the correct interval in $\mathcal{B}_0$ and $\mathcal{B}_1$.

Algorithm 4: DETERMINEINTERVAL$(x_1, \ldots, x_n)$.

$A_1, A_2, \ldots, A_k$ are disjoint intervals of size $\approx 2t\sqrt{n}$ covering $[0, n]$, with t chosen later.
Let A_i be $\varnothing$ if $i < 1$ or $i > k$.
$B_i := A_i \cup A_{i+1} \cup A_{i+2}$.
$\mathcal{B}_s := \{B_i : i \equiv s \bmod 3\}$.
Enc is the encoding function of a code with relative distance 0.25 and constant rate K.

---

1. Transmit x_i
2. Compute $h_i := \frac{\text{number of 1's received}}{1-p}$
3. For $s = 0, 1, 2$:
   a. Find interval in $\mathcal{B}_s$ containing h_i, called I. I is encoded as a string s_I (of size $O(\log n)$).
   b. Let $i' = i \bmod \log n$ and transmit $\mathsf{Enc}(s_I)_{[K(i'-1)+1, Ki']}$ over K rounds
   c. $C_{i,s} := \begin{cases} s_I & \text{if at least } .88K \log n \text{ bits were received in Step 3b} \\ \text{decoded string} & \text{if fewer than } .88K \log n \text{ bits were received in Step 3b} \end{cases}$
4. Return $C_{i,0}, C_{i,1}$ and $C_{i,2}$.

And assuming at least 0.95 fraction of the processors decode the correct intervals in $\mathcal{B}_0$ and $\mathcal{B}_1$, we can show once again using Chernoff bounds and union bound, that the number of bits from the encoded string of the correct interval received by each processor is more than $0.9Kn$ with probability at least $1 - \exp(-\Omega(n))$, which means with exponentially high probability, every processor decodes the correct interval in $\mathcal{B}_0$ and $\mathcal{B}_1$. ◀

For the second step, our goal is the following: given that every processor knows an interval $[a, b]$ in which the Hamming weight of the input string lies, it can recover the value of the Hamming weight in $O(1)$ rounds.

▶ **Lemma 10.** *On running Algorithm 5, all processors return the Hamming weight s of X with probability at least* 0.9.

Proof. Define $\widehat{\theta}_s$ to be the fraction of β_i transmitted in Step 3 that are 1.

We can lower bound $\theta_{\ell+1} - \theta_\ell$ for $x \leq \ell < y$ by $\frac{c}{\sqrt{n}}$ where c is some constant [11, Lemma 41]. The probability that $|\theta_s - \widehat{\theta}_s|$ is at most $\frac{c}{8\sqrt{n}}$ can be made at least 0.99 with an appropriate choice of the number of repetitions γ. Similarly, we can ensure that $|\widehat{\theta}_s - \widehat{\theta}_{s,i}|$ is at most $\frac{c}{8\sqrt{n}}$ with probability at least 0.99.

By Chernoff bounds, the fraction of processors for which $|\widehat{\theta}_s - \widehat{\theta}_{s,i}| < \frac{c}{8\sqrt{n}}$ is at least 0.95 with probability at least $1 - \exp(-\Omega(n))$. Thus, conditioned on $|\theta_s - \widehat{\theta}_s| < \frac{c}{8\sqrt{n}}$, we have that for at least 0.95 of the processors, $|\theta_s - \widehat{\theta}_{s,i}| < \frac{c}{4\sqrt{n}}$. Further, the string $S_1 S_2 \ldots S_n$ transmitted in Step 6 with random erasures has distance less than the decoding radius of $\mathcal{C}$ of $\mathsf{Enc}(s)$ with probability at least $1 - \exp(-\Omega(n))$, in which case all processors can correctly output s.

Since the condition $|\theta_s - \widehat{\theta}_s| < \frac{c}{8\sqrt{n}}$ holds with probability at least 0.99, the required guarantees of the Lemma hold. ◀

Proof of Theorem 8. The processors run Algorithm 4 to obtain 3 candidate intervals I_1, I_2 and I_3, and with exponentially high probability, at least two of these candidate intervals

Algorithm 5: PINPOINTWEIGHT$(x_1, x_2, \ldots, x_n; [a, b])$.

$[a, b]$ is the interval of length up to $3\sqrt{n}$ where the Hamming weight is promised to lie
Enc is the encoding function of a code $\mathcal{C}$ with relative distance 0.25 that maps $\log n$ bit strings to $K \log n$ bit strings
Let θ_s be defined as the probability that when flipping s coins, each coming up heads with probability $1 - p$, at least $(1 - p)\left(\frac{a+b}{2}\right)$ come up heads.

---

1. Transmit x_i
   Let Y be the number of 1's received.
2. $\beta_i := \begin{cases} 1 & \text{if number of 1's received is greater than } (1-p)\left(\frac{a+b}{2}\right) \\ 0 & \text{otherwise} \end{cases}$
3. Transmit β_i
4. Let $\widehat{\theta}_{s,i}$ be the fraction of received bits from Step 3 that are 1 (i.e., the total number of 1's received, divided by the total number of 1's or 0's received).
5. $\widehat{s}_i = \arg\min_\ell |\theta_\ell - \widehat{\theta}_{s,i}|$
6. Let $i' = i \bmod \log n$ and transmit $\mathsf{Enc}(\widehat{s}_i)_{[K(i'-1)+1, Ki']}$ over K rounds
7. $\tilde{s}_i = \begin{cases} \text{decoded string} & \text{if at least } .88K \log n \text{ bits were received in Step 6} \\ \widehat{s}_i & \text{if fewer than } .88K \log n \text{ bits were received in Step 6} \end{cases}$

contain the Hamming weight. The processors run Algorithm 5 on each of the three intervals and processor i obtains outputs n_0, n_1 and n_2 respectively. With constant probability, at least two of n_0, n_1 and n_2 are the same and equal to the correct Hamming weight, and hence outputting the majority of the three matches the guarantee. ◀