Brief Announcement: Multiparty Interactive Communication with Private Channels

Abhinav Aggarwal
University of New Mexico
Albuquerque, NM 87131
abhiag@unm.edu

Varsha Dani
University of New Mexico
Albuquerque, NM 87131
varsha@unm.edu

Thomas P. Hayes
University of New Mexico
Albuquerque, NM 87131
hayes@cs.unm.edu

Jared Saia
University of New Mexico
Albuquerque, NM 87131
saia@cs.unm.edu

ABSTRACT

A group of n players wants to run a distributed protocol $\mathcal{P}$ over a network where communication occurs via private point-to-point channels. Can we efficiently simulate $\mathcal{P}$ in the presence of an adversary who knows $\mathcal{P}$ and is able to maliciously flip bits on the channels? We show that this is possible, even when L, the number of bits sent in $\mathcal{P}$, the average message size α in $\mathcal{P}$, and T, the number of bits flipped by the adversary are not known in advance. In particular, we show how to create a robust version of $\mathcal{P}$, $\mathcal{P}'$ such that 1) $\mathcal{P}'$ fails with probability at most δ, for any $\delta > 0$; and 2) $\mathcal{P}'$ sends $O(L(1 + (1/\alpha)\log(nL/\delta)) + T)$ bits. We note that if α is $\Omega(\log(nL/\delta))$, then $\mathcal{P}'$ sends only $O(L + T)$ bits, and is therefore within a constant factor of optimal. Critically, our result requires that $\mathcal{P}$ runs correctly in an asynchronous network and our protocol $\mathcal{P}'$ must run in a synchronous network.

KEYWORDS

Interactive Communication, Private Channels, Resource Competitive Analysis

ACM Reference format:
Abhinav Aggarwal, Varsha Dani, Thomas P. Hayes, and Jared Saia. 2019. Brief Announcement: Multiparty Interactive Communication with Private Channels. In *Proceedings of 2019 ACM Symposium on Principles of Distributed Computing, Toronto, ON, Canada, July 29-August 2, 2019 (PODC '19),* 3 pages.
DOI: 10.1145/3293611.3331571

1 INTRODUCTION

How can we compute in the presence of noise? The problem of interactive communication formalizes this problem [2, 9, 10]. In this problem, n players are in a network connected by point-to-point binary symmetric channels, and we want to simulate a distributed protocol $\mathcal{P}$, even when an adversary can flip bits on any of the channels at any time. Our goal is to create a new protocol $\mathcal{P}'$ that correctly simulates $\mathcal{P}$ even in the presence of such adversarial bit flipping, and does so while minimizing the number of bits sent.

PODC '19, Toronto, ON, Canada
 978-1-4503-6217-7/19/07…$15.00
DOI: 10.1145/3293611.3331571

Under certain special conditions on $\mathcal{P}$, it is already known how to create a $\mathcal{P}'$ with a multiplicative blowup in bits sent that is $O(\log n)$ [9], or even $O(1)$ for certain network topologies [2, 6, 7]. However, the conditions on $\mathcal{P}$ are strong: it must be a protocol that sends exactly 1 bit over every channel in every time step. Unfortunately, in a complete network, where the average number of messages sent per round in $\mathcal{P}$ is only $\tilde{O}(n)$, the multiplicative blowup can become $\tilde{\Omega}(n)$. This is much too high for practitioners who need noise-tolerant distributed algorithms.

A naive idea is to run the 2-player algorithm of Dani et al. [5] over each communication channel. Unfortunately, this fails. In particular, the protocol of [5] assumes that a player sends a bit on the channel in each time step[1]. However, in the multiparty problem, there may be many time steps where, for many channels, neither player on the channel sends a bit. This can happen because delay due to noise on one channel is holding up the protocol on all other channels, or simply because both players connected by a channel are waiting for messages from other players before using that channel.

In this paper, we take a first step toward designing an algorithm with costs that scale well for any network topology. In particular, if an adversary flips T bits, then our algorithm sends $\tilde{O}(L + T)$ bits, where L is the number of bits sent in $\mathcal{P}$. Here the $\tilde{O}$ notation hides a logarithmic term in n, L and δ; if the desired δ is polynomially (or even sub-polynomially) small in n, then the $\tilde{O}$ notation hides a logarithmic (polylogarithmic) term just in n and L. Moreover, if the average message size in $\mathcal{P}$ is $\Omega(\log(nL/\delta))$, then $\mathcal{P}'$ sends only $O(L + T)$ bits which is within a constant factor of optimal. Importantly, our algorithm requires no *a priori* knowledge of T or L (unlike all prior interactive communication results).

There exist many examples of asynchronous algorithms known for common problems that need to be solved in noisy networks. Our result allows using any asynchronous algorithm to design a new synchronous algorithm that is robust to channel noise. This new algorithm sends significantly fewer bits compared to prior work, which sends $\Omega(nL)$ bits [2, 3, 6, 9]. Additionally, when $T = \tilde{O}(L)$, the new algorithm sends only $\tilde{O}(L)$ bits.

Model Details: To obtain our result, we make two key simplifying assumptions over [2, 3, 6, 9]. First, as in [5], we assume all

[1] A time step is defined as the amount of time that it takes to send one bit over a channel.

channels are private, but otherwise make no cryptographic assumptions. Second, we assume that $\mathcal{P}$ is asynchronous: it runs correctly even with arbitrary message delays. As in previous work, $\mathcal{P}'$ is assumed to run in a synchronous network.

In each time step, a bit value is set for every channel. This is the value received by any player listening on the channel. In any contiguous sequence of time steps where no player sends a bit, the adversary sets the bits, and pays a cost of 1 for each time it changes the value. As in prior work [2, 9], we assume that all players know the value n but may not know the values L, T and α. In the case where we want an error probability that is polynomially small in n, we require that all players know a desired error probability δ.

We make use of two key ideas from [5]. First, we use *Algebraic Manipulation Detection (AMD) Codes* [4]. These codes enable detection of any bit flips in a code word with probability of error that is exponentially small in the number of bits added to the message word.[2] Second, similarly to [5], we use error-correcting encoding of AMD code words to ensure that when we have to resend a message, our costs are linear in the number of bits flipped by the adversary.

How is our model different? Our work is not directly comparable to most past results because we do not quantify our results in terms of the fraction of messages that are corrupted, and more importantly, we assume that communication takes place over private channels. The stronger assumption of private channels means that up to a $1/\log(nL)$ fraction of message bits can be corrupted, and our algorithm is still likely to succeed with a cost overhead that is only $O(\log(nL))$. In our setting, the adversarial strategy of trying to cut off all communications to and from a single player requires corruption of much more than a $1/n$ fraction of bits. This is true since our protocol can detect the noise, and then increase the fraction of the total communication involving the beleaguered player. Finally, to the best of our knowledge, all prior work on Interactive Communication assumes players know L in advance. We are able to remove this assumption of a priori knowledge of L.

2 ALGORITHM OVERVIEW

Our algorithm proceeds in rounds, each of which consists of the following steps.[3]

(1) If u has a message for v, it initiates a message exchange by asking v for a key.
(2) Upon receipt of this key, u sends the message along with the key.
(3) v terminates the message exchange upon successful authentication and retrieval of the message.
(4) u terminates the message exchange upon hearing silence from v.

This goes on until all the messages in $\mathcal{P}$ have been communicated to the intended recipients.

Rounds. For each message m in $\mathcal{P}$ that needs to be sent from some player u to his neighbor v, we communicate m through a sequence of exchanges between players u, referred to as *Alice*, and v, referred to as *Bob*, in $\mathcal{P}'$ using Algorithms 1 and 2, respectively. The sequence of time steps corresponding to steps 1-4 of the algorithm overview constitute what we call a *round*. Thus, each round of $\mathcal{P}$ consists of exactly four *words*, one for each of the steps 1-4. The length of each word in round r is denoted w_r. This depends on the round number r and is therefore a function of the current time step, which can be computed independently by each player using the global clock. We make w_r gradually increase with r (the exact dependence will be provided shortly).

Since each message in $\mathcal{P}$ is just a single bit, it takes exactly one round to be communicated in $\mathcal{P}'$, if no successful corruption happens, and more otherwise. If a round for some message m is corrupted, we attempt resending m in the subsequent round, which provides higher security due to a larger word length.

At the beginning of each round, Bob must listen for a key request during the first w_r time steps. If no valid request is received, he idles until the start of the next round. Thus Bob is active in every round. For her part, Alice only participates in a round if, in $\mathcal{P}$, she has a message to send to Bob.

Parity Bit. Observe that Alice and Bob may have different views on whether a particular round was successful. If Bob encounters two rounds that contain the same message, with no silent round in between, he needs to distinguish between whether Alice is resending the message or whether Alice's next message happens to be the same as the previous one. To disambiguate these cases, Alice appends a bit b to each message m that she sends to Bob, where b is the parity of the index of m.

Word Generation. Both Alice and Bob generate their words for round r using a function $\mathscr{E}_r(x, k)$, described below, which returns the encoding of the word's content x using the key k based on the security settings for round r. Here x may be the message m from $\mathcal{P}$, a special keyword used by Alice to request Bob's key, or Bob's key for the round. The key k, for round r, is a string of length $2\left\lceil \log \frac{4n\pi r}{\sqrt{\delta}} \right\rceil$ bits. This key length, and other corresponding encoding parameters, are chosen to ensure the failure events for our algorithm occur within the given error tolerance, δ (see [1] for more details).

In Alice's first call to $\mathscr{E}$, the key k is a random string. In Bob's call, k is the key he received from Alice in the previous word, but x is a random string of the appropriate length. In Alice's second call, k is the key she received from Bob in the previous word. Alice and Bob generate fresh random keys for each round in which they desire to send a message.

The function $\mathscr{E}_r(x, k)$ is formally defined as follows. The function uses a key k of length $\kappa_r = 2\left\lceil \log \frac{4n\pi r}{\sqrt{\delta}} \right\rceil$ bits and appends it to the content x. Then, it encodes this concatenated string of bits with an AMD code using $\eta_r = \frac{\delta}{2n^2\pi^2r^2}$. Then, it encodes this AMD codeword with a (1/3)-error-correcting code. Finally, it pads $38\left\lceil \log \frac{2n\pi r}{\sqrt{\delta}} \right\rceil$ random bits to the ECC codeword. These random bits are added to ensure that players are unlikely to confuse words with silence and that the adversary is unlikely to flip bits of a word to forge silence. As a result, the word length for round r is given as $w_r = O\left(\log \frac{nr}{\delta}\right)$.

Similar to $\mathscr{E}_r(x, k)$, we define a decoding function which returns either a pair (x', k') such that $\mathscr{E}_r(x', k') = m$ or returns $(\perp, \perp)$, if

[2] We note that this error probability is smaller than what can be achieved via digital signatures [8].

[3] For convenience, our presentation assumes two bidirectional channels between each pair of (neighboring) players, one for each player to initiate a round with the other. We can simulate these two channels by multiplexing over a single channel, at the cost of a factor 2 increase in the number of time steps.

no such (x', k') exists. It begins by stripping off the padding at the end of m to obtain a shorter string m'. Then it decodes the error correction, computing $m'' = \texttt{ecDec}(m')$. If $\texttt{IsCodeword}(m'', \eta_r)$, then $\mathscr{D}$ outputs $\texttt{amdDec}(m'', \eta)$, otherwise it returns $(\perp, \perp)$.

Failure Events. There are certain events which may cause our algorithm to fail. More specifically, it is possible that the adversary (1) converts one AMD codeword to another, so that the decoded content is different from the content intended; (2) converts a non-silence word into silence; and (3) correctly guesses some player's key and uses it to communicate bits that are not in $\mathcal{P}$. We bound the probability of these failure events by the error tolerance, δ.

3 OVERVIEW OF OUR RESULTS

Our main result is summarized in the following theorem.

Theorem 3.1. *Let $\mathcal{P}$ be an asynchronous protocol for $n \geq 2$ players that sends a total of L bits. Let α be the average message length in $\mathcal{P}$. Let $\delta > 0$. Then we can create a synchronous protocol $\mathcal{P}'$ such that for any number T of adversarial bit flips, $\mathcal{P}'$ succeeds with probability at least $1 - \delta$ and sends $O\left(L\left(1 + \frac{1}{\alpha}\log\left(\frac{nL}{\delta}\right)\right) + T\right)$ bits. The total latency of $\mathcal{P}'$ is $O\left(\max_p\left\{\Lambda_p\left(1 + \frac{1}{\alpha}\log\left(\frac{n(L+T)}{\delta}\right)\right) + T_p\right\}\right)$ where p ranges over all communication paths in the asynchronous simulation of $\mathcal{P}$, Λ_p is the latency of p, and T_p is the total number of bits flipped by the adversary on channels in p.*

To compare our result with other work, we compute the coding rate of our algorithm. Recall that the coding rate of an interactive communication algorithm is the number of bits sent in $\mathcal{P}'$ divided by the number of bits sent in $\mathcal{P}$. In our work we do not assume that the noise rate is known in advance, but only require that the adversary flip a finite number of bits. Thus, we compute the coding rate as a function of the *a posteriori* noise rate. Such a comparison is only meaningful when the adversary's total budget is less than L, the length of $\mathcal{P}$, so for this section we will assume that $T < L$.

Let L' denote the length of $\mathcal{P}'$. Theorem 3.1 states that our algorithm achieves $L' \leq CL\left(1 + \frac{1}{\alpha}\log\left(\frac{nL}{\delta}\right)\right) + CT$ for some constant C, where δ is the permissible failure probability for the algorithm and α is the average message length in $\mathcal{P}$. Furthermore, since $T < L$ this translates to the absolute upper bound $L' \leq CL\left(2 + \frac{1}{\alpha}\log\left(\frac{nL}{\delta}\right)\right)$. Let $\varepsilon = T/L'$ be the *a posteriori* noise rate. Then $T = \varepsilon L'$. Making this substitution for T in the inequality above and dividing by L, we get $\frac{L'}{L} \leq 2C + \frac{C}{\alpha}\left\lceil \log(nL/\delta) + \log\left(1 + \varepsilon\left(2C + \frac{C\log(2nL/\delta)}{\alpha}\right)\right)\right\rceil$. Finally, using $\log(1+x) \leq x$ on the second logarithm in this expression, we get $\frac{L'}{L} \leq \left(2C + \frac{C\log(2nL/\delta)}{\alpha}\right)\left(1 + \frac{\varepsilon C}{\alpha}\right)$.

To summarize, for the worst case where $\alpha = 1$, the above shows that we achieve a coding rate that increases linearly with the noise rate ε and with the logarithm of nL/δ. In particular, the coding rate is $O((1+\varepsilon)\log(nL/\delta))$. For arbitrary α, we achieve a coding rate of $O((1+\varepsilon/\alpha)\log(nL/\delta)/\alpha)$.

Algorithm 1: Sender's algorithm.

1. Generate a random session key k_A.
2. Request a session key k_B from the recipient. Include k_A in the message;
3. If silence is heard, wait for $2w_r$ time steps and proceed to the next round;
4. Else, decode the message and check if k_A is correctly reproduced. If so, append the message m with k_B and send it across.
5. Proceed to the next message if silence is heard, else resend the message.

Algorithm 2: Receiver's algorithm.

1. Generate a random session key k_B.
2. If a valid key request is heard, send k_B. Echo k_A in the message.
3. If silence is heard, wait for $3w_r$ time steps and proceed to the next round;
4. Else, decode the message and check if k_B is correctly reproduced. If so, extract the message m and stay silent. Else, send noise for w_r time steps.

ACKNOWLEDGMENTS

This work is supported by the National Science Foundation grants CNS 1816250, CCF 1320994, CNS 1318880 and CCF 1613772.

REFERENCES

[1] Abhinav Aggarwal, Varsha Dani, Thomas P. Hayes, and Jared Saia. 2018. Multiparty Interactive Communication with Private Channels. (2018). https://goo.gl/JZqPWe

[2] Noga Alon, Mark Braverman, Klim Efremenko, Ran Gelles, and Bernhard Haeupler. 2016. Reliable Communication Over Highly Connected Noisy Networks. In *Proceedings of the ACM Symposium on Principles of Distributed Computing (PODC)*. ACM, 165–173.

[3] 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, January 11-14, 2018, Cambridge, MA, USA*. 50:1–50:20.

[4] Ronald Cramer, Yevgeniy Dodis, Serge Fehr, Carles Padró, and Daniel Wichs. 2008. Detection of Algebraic Manipulation with Applications to Robust Secret Sharing and Fuzzy Extractors. In *Advances in Cryptology–EUROCRYPT 2008*. Springer, 471–488.

[5] Varsha Dani, Thomas P. Hayes, Mahnush Movahedi, Jared Saia, and Maxwell Young. 2018. Interactive Communication with Unknown Noise Rate. *Information and Computation* (2018).

[6] 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*. SIAM, 240–258.

[7] Abhishek Jain, Yael Tauman Kalai, and Allison Bishop Lewko. 2015. Interactive coding for multiparty protocols. In *Proceedings of the 2015 Conference on Innovations in Theoretical Computer Science*. ACM, 1–10.

[8] Jonathan Katz and Yehuda Lindell. 2014. *Introduction to modern cryptography*. CRC press.

[9] Sridhar Rajagopalan and Leonard Schulman. 1994. A Coding Theorem for Distributed Computation. In *Proceedings of ACM Symposium on Theory of computing (STOC)*. ACM, 790–799.

[10] Leonard J. Schulman. 1996. Coding for Interactive Communication. *IEEE Transactions on Information Theory* 42, 6 (1996), 1745–1756.