Ordered and Delayed Adversaries
and How to Work against Them on Shared Channel

Marek Klonowski∗ Dariusz R. Kowalski† Jarosław Mirek†

Abstract

Performance of a distributed algorithm depends on environment in which the algorithm is executed. This is often modeled as a game between the algorithm and a conceptual adversary causing specific distractions. In this work we define a class of ordered adversaries, which cause distractions according to some partial order fixed by the adversary before the execution, and study how they affect performance of algorithms. For this purpose, we focus on well-known Do-All problem of performing \( t \) tasks on a shared channel consisting of \( p \) crash-prone stations. The channel restricts communication by the fact that no message is delivered to the alive stations if more than one station transmits at the same time. The most popular and meaningful performance measure for Do-All type of problems considered in the literature is work, defined as the total number of available processor steps during the whole execution.

The question addressed in this work is how the ordered adversaries controlling crashes of stations influence work performance of Do-All algorithms. We provide two randomized distributed algorithms, whose work performance is analyzed with high probability against the three representative ordered adversaries generating stations’ crashes. The first algorithm solves the Do-All problem with work \( O(t + p\sqrt{t} \log p) \) against the Linearly-Ordered adversary, which is restricted by some pre-defined linear order of (potentially) crashing stations. Surprisingly, the upper bound on performance of this algorithm does not depend on the number of crashes \( f \) and is close to the absolute lower bound \( \Omega(t + p\sqrt{t}) \) proved in [13]. Another algorithm is developed against the Weakly-Adaptive adversary, which is restricted by some pre-defined set of crash-prone stations; it can be seen as an ordered adversary with the order being an anti-chain consisting of (potentially) crashing stations. The work done by this algorithm is \( O(t + p\sqrt{t} + p \min \{ p/(p-f), t \} \log p) \), which is close to the lower bound \( \Omega(t + p\sqrt{t} + p \min \{ p/(p-f), t \} ) \) proved in [13]. We generalize this result to the class of adversaries restricted by partial order of \( f \) stations with maximum anti-chain of size \( k \), in which case the work of the algorithm is bounded by \( O(t + p\sqrt{t} + p \min \{ p/(p-f), k, t \} \log p) \). We complement this result by almost matching lower bound \( \Omega(t + p\sqrt{t} + p \min \{ p/(p-f), k, t \} ) \).

Additionally to the results for the ordered adversaries, we consider a class of delayed adaptive adversaries, that is, who could see random choices with some delay. We present an algorithm that works efficiently against the 1-RD adversary, which could see random choices of stations with one round delay, achieving close to optimal \( O(t + p\sqrt{t} \log^2 p) \) work complexity. This shows that restricting adversary by even one round delay results in (almost) optimal work performance on a shared channel, regardless of the order of crashes.

Keywords: Performing tasks, Do-All, Shared channel, Multiple-access channel, Ordered adversaries, Delayed adversaries, Crash failures, Distributed algorithms, Randomized Algorithms, Work complexity, Time complexity, Transmission energy complexity.

∗Faculty of Fundamental Problems of Technology, Wrocław University of Technology, Wybrzeże Wyspiańskiego 27, 50-370 Wrocław, Poland. Email: Marek.Klonowski@pwr.edu.pl
†Department of Computer Science, University of Liverpool, Ashton Building, Ashton Street, Liverpool L69 3BX, UK. Email: {D.Kowalski, J.Mirek}@liverpool.ac.uk.

Partially supported by Polish National Science Center grant 2015/17/B/ST6/01897.
1 Introduction

We consider the problem of performing \( t \) similar and independent tasks in a distributed system prone to processor crashes. This problem, called \textit{Do-All}, was introduced by Dwork et al. \cite{dwork1988} in the context of a message-passing system. Over the years the \textit{Do-All} problem became a pillar of distributed computing and has been studied widely from different perspectives \cite{24}. It became customary that the bottleneck for the problem is often connected with the scheduler.

The distributed system studied in our paper is based on communication over a shared channel, also called a multiple-access channel, and was first studied in the context of \textit{Do-All} problem by Chlebus et al. \cite{13}. In this work we adopt the model from that paper.

The channel is synchronous and consists of \( p \) processors, also called stations, prone to crashes. It provides a global clock, which defines the same rounds for all alive stations. A message sent by a station at a round is received by all alive stations only if it is the only transmitter in this round; we call such a transmission successful. Otherwise, unless stated differently, we assume that no station receives any meaningful feedback from the channel medium, except an acknowledgment of its successful transmission.

It is worth emphasizing that the communication channel in our model can easily be made resistant to non-synchronized processor clocks, using methods developed previously \cite{26}.

Stations are prone to crash failures. Allowable patterns of failures are determined by abstract adversarial models. Historically, main distinction is between \textit{adaptive} and \textit{ooblusive} adversaries; the former can make decisions during the computation while the latter cannot. Another characteristic of an adversary is being \textit{size-bounded}, or more specifically \textit{\( f \)-bounded}, if it may fail at most \( f \) stations, for a parameter \( 0 \leq f < p \); a \textit{linearly-bounded} adversary is simply a \( c \cdot p \)-bounded adversary, for some constant \( 0 < c < 1 \). We introduce the notion of \textit{ordered adaptive adversary}, or simply \textit{ordered adversary}, who can crash stations according to some pre-selected order (unknown to the algorithm). On the other hand, a \textit{strongly-adaptive} adversary is not restricted by any constraint other than being \( f \)-bounded for some \( f < p \).

Adversaries described by a partial order are interesting on their own right. Nevertheless, to the best of our knowledge, such adversaries were not considered in literature so far and hence form novel scenarios for testing and having better understanding of certain distributed algorithms. Furthermore, our results prove that according to the type of the partial order, algorithms need different design and have different complexity.

In view of crashes and specific nature of \textit{Do-All} problem, the most meaningful measure considered in the literature is \textit{work}, accounting the total number of available processor steps in the computation. We require from algorithms to be \textit{reliable} in the sense that they must perform all the tasks for any pattern of crashes such that at least one station remains operational in an execution. Chlebus et al.\cite{13} showed that \( \Omega(t + p\sqrt{t}) \) work is inevitable for any reliable algorithm, even randomized, even for the channel with collision detection (i.e., when alive stations can recognize no transmission from at least two transmitting stations in a round), and even if no failure occurs. This is the absolute lower bound on the work complexity of the \textit{Do-All} problem on a shared channel. On the other hand, it is known that this bound is achieved by a deterministic algorithm for channels with enhanced feedback, such as collision detection or beeping, cf., \cite{13}, and therefore such models are not challenging any more from perspective of reliable task performance. \textit{Our goal is to check how different classes of adversaries, especially those constrained by a partial order of crashes, influence work performance of Do-All algorithms on a simple shared channel with acknowledgments only.}

1.1 Previous work

The \textit{Do-All} problem was introduced by Dwork, Halpern and Waarts \cite{19} in the context of a message-passing model with processor crashes.

Chlebus, Kowalski and Lingas \cite{13} were the first who considered \textit{Do-All} in a multiple-access channel. Apart from the absolute lower bound for work complexity, discussed earlier, they also showed
a deterministic algorithm matching this performance in case of channel with collision detection. Regarding the channel without collision detection, they developed a deterministic solution that is optimal for such weak channel with respect to the lower bound they proved regarding the channel without collision detection, they developed a deterministic solution that is optimal.

The lower bound holds also for randomized algorithms against the strongly adaptive adversary, that is, the adversary who can see random choices and react online, which shows that randomization does not help against strongly adaptive adversary.

Furthermore, the paper contains a randomized solution that is efficient against a weakly adaptive adversary who can fail only a constant fraction of stations. A weakly adaptive adversary is such that she needs to select \( f \) crash-prone processors in advance, based only on the knowledge of algorithm but without any knowledge of random bits; then, during the execution, she can fail only processors from that set. This algorithm matches the absolute lower bound on work. If the adversary is not linearly bounded, that is, \( f < p \) could be arbitrary, then they only proved a lower bound of \( \Omega\left(t + \frac{p}{p^*}t + p \min \left\{ \frac{p}{p^*}, t \right\} \right) \).

Clementi, Monti and Silvestri [16] investigated Do-All in the communication model of a multiple-access channel without collision detection. It studied \( F \)-reliable protocols, which are correct if the number of crashes is at most \( F \), for a parameter \( F < p \). They obtained tight bounds on the time and work of \( F \)-reliable deterministic protocols. In particular, the bound on work shown in [16] is \( \Theta(t + F \cdot \min(t, F)) \). In this paper, we consider protocols that are correct for any number of crashes smaller than \( p \), which is the same as \( (p - 1) \)-reliability. Moreover, the complexity bounds of our algorithms, for the channel without collision detection, are parametrized by the number \( f \) of crashes that actually occur in an execution. Results shown in [16] were also investigating the time perspective with a lower bound on time complexity equal \( \Omega\left(\frac{t}{2p} + \min \left\{ \frac{t}{p}, F + \sqrt{t} \right\} \right) \). However, the protocols make explicit use of the knowledge of \( F \). In this paper we give some remarks on time and energy complexity, but those statements are correct for an arbitrary \( f \).

1.2 Related work

Do-All problem. After the seminal work by Dwork, Halpern and Waarts [19], the Do-All problem was studied in a number of follow-up papers [8, 9, 12, 17, 20] in the context of a message-passing model, in which every node can send a message to any subset of nodes in one round. Dwork et al. [19] analyzed task-oriented work, in which each performance of a task contributes a unit to complexity, and the communication complexity defined as the number of point-to-point messages. De Prisco, Mayer and Yung [17] were the first to use the available processor steps [31] as the measure of work for solutions of Do-All. They developed an algorithm which has work \( O(t + (f + 1)p) \) and message complexity \( O((f + 1)p) \). Galil, Mayer and Yung [20] improved the message complexity to \( O(fp^2 + \min\{f + 1, \log p\}p) \), for any positive \( \varepsilon \), while maintaining the same work complexity. This was achieved as a by-product of their investigation of the Byzantine agreement with crash failures, for which they found a message-efficient solution. Chlebus, De Prisco and Shvartsman [8] studied failure models allowing restarts. Chlebus and Kowalski [12] studied the Do-All problem when occurrences of failures are controlled by the weakly-adaptive linearly-bounded adversary. They developed a randomized algorithm with the expected effort \( O(p \log p) \), in the case \( p = t \), which is asymptotically smaller than the lower bound \( \Omega(p \log p / \log \log p) \) on work of any deterministic algorithm. Chlebus, Gasieniec, Kowalski and Shvartsman [9] developed a deterministic algorithm with effort \( O(t + p^2) \), for some specific constant \( a \), where \( 1 < a < 2 \), against the unbounded adversary, which is the first algorithm with the property that both work and communication are \( o(t + p^2) \) against this adversary. They also gave an algorithm achieving both work and communication \( O(t + p \log^2 p) \) against a strongly-adaptive linearly-bounded adversary. All the previously known deterministic algorithms had either work or communication performance \( \Omega(t + p^2) \) when as many as a linear fraction of processing units could be failed by a strongly-adaptive adversary. Georgiou, Kowalski and Shvartsman [23] developed an algorithm with work \( O(t + p^{1+\varepsilon}) \), for any fixed constant \( \varepsilon \), by an approach based on gossiping. Kowalski and Shvartsman in [36] studied Do-All in an asynchronous message-passing mode when executions are restricted such that every message delay is at most \( d \). They showed lower bound \( \Omega(t + pd \log_d p) \) on the expected
work. They developed several algorithms, among them a deterministic one with work $O((t + pd) \log p)$. For further developments we refer the reader to the book by Georgiou and Shvartsman [24].

**Related problems on a shared channel.** Most of work in this model focused on communication problems, see the surveys [7][21]. Among the most popular protocols for resolving contention on the channel are Aloha [1] and exponential backoff [39]. The two most related research problems are as follows.

The **selection problem** is about how to have an input message broadcast successfully if only some among the stations hold input messages while the other do not. Willard [43] developed protocols solving this problem in the expected time $O(\log \log n)$ in the channel with collision detection. Kushilevitz and Mansour [37] showed a lower bound $\Omega(\log n)$ for this problem in case of a lack of collision detection, what explains the exponential gap between this model and the one with collision detection. Martel [38] studied the related problem of finding maximum within the values stored by a group of stations.

The **wake-up** problem is about how to perform a successful broadcast as quickly as possible after a start of a system, in a scenario when the spontaneous times to join execution are independent over all stations and are controlled by an adversary. Gąsieniec, Pelc and Peleg [22] introduced this problem for the multiple-access channel and examined different synchrony modes confronted with the complexity of solutions. Mainly, they showed that if the system has access to a global clock, then wake-up can be achieved in $O(\log n)$ expected time by a randomized algorithm. On the other hand, for stations that are not synchronized, there is a randomized solution working in $O(\log n)$ expected time. This result was however improved by Jurdziński and Stachowiak [30] with a protocol disregarding the global clock, working in the expected time $O(\log n)$. Results in [22] stated that $\Omega(n)$ is the lower bound for deterministic protocols, as well as that there is a deterministic protocol working in time $O(n \log^2 n)$.

Deterministic solutions for the wake-up problem are based on radio synchronizers which formal definition is as follows: a binary $n \times m$ array $S$ is a $(n, k)$-synchronizer of length $m$ if for any nonempty set $A \subseteq \{1, \ldots, n\}$ of at most $k$ rows and for arbitrary shifts of rows in $A$, each by a distance at most $m$, there is a column with an occurrence of 1 in exactly one shifted row in $A$. Chrobak, Gąsieniec and Kowalski [15] introduced such synchronizers and used them for wake-up, leader election and synchronization of local clocks in multi-hop radio networks, although these structures were implicitly used before in [22] to obtain weaker results. It was shown in [15] that such structures exist with $m = O(k^2 \log n)$, for given $n$ and $k$. Indyk [28] showed that synchronizers with $k = n$ and $m = O(n^{1+\varepsilon})$ can be constructed in time $O(2^{\text{polylog } n})$, for any constant $\varepsilon > 0$. Chlebus and Kowalski [11] proved that $(n, k)$-synchronizers of length $O(k^2 \text{ polylog } n)$ can be constructed in time polynomial in $n$.

In [32] Klonowski, Kutyłowski and Zatopiański proposed randomized solution for wake-up problem with polylogarithmic execution time and sublogarithmic energy complexity (maximal number of transmissions over all stations) with respect to the number of stations.

Jurdziński, Kutyłowski and Zatopiański [29] considered the, closely related to the selection and contention resolution problems, leader election problem for the channel without collision detection. giving a deterministic algorithm with sub-logarithmic energy cost. They also proved doubly-logarithmic lower bound for the problem.

The **contention resolution** problem, in which a subset of some $k$ among all $n$ stations have messages, and all these messages need to be transmitted successfully on the channel as quickly as possible. Komlós and Greenberg [34] proposed a deterministic solution allowing to achieve this in time $O(k + k \log(n/k))$, where $n$ and $k$ are known. Kowalski [35] gave an explicit solution of complexity $O(k \text{ polylog } n)$, while the lower bound $\Omega(k(\log n)/(\log k))$ was shown by Greenberg and Winograd [25]. The work by Chlebus, Gołąb and Kowalski [10] regarded broadcasting spanning forests on a multiple-access channel, with locally stored edges of an input graph.

Significant part of recent results on the communication model considered in literature is focused on jamming-resistant protocols motivated by applications in single-hop wireless networks. To the best of our knowledge this vein of research was initiated in [3] by Awerbuch, Richa and Scheideler, wherein authors introduced a model of adversary capable of jamming up to $(1 - \varepsilon)$ of the time steps (slots). The following papers [40][41] by Richa, Scheideler, Schmid and Zhang proposed several algorithms that can
reinforce the communication even for a very strong, adaptive adversary. For the same model Klonowski and Pająk in [33] proposed an optimal leader election protocol, using a different algorithmic approach.

The similar model of a jamming adversary was considered by Bender, Fineman, Gibert and Young in [4]. Authors consider a modified, robust exponential backoff protocol that require $O(\log^2 n + T)$ attempts to the channel if there are at most $T$ jammed slots. Motivated by saving energy, authors try to find maximal global throughput while reducing device costs expressed by the number of attendants in the channel.

Finally, there are several recent results on finding very exact approximations of the network. In [5] Brandes, Kardas, Klonowski, Pająk and Wattenhofer proposed an algorithm for the network of $n$ stations that returns $(1 + \varepsilon)$-approximation of $n$ with probability at least $1 - 1/f$. This procedure takes $O(\log \log n + \log f/\varepsilon^2)$ time slots. This result was also proved to be time-optimal. In [6] Chen, Zhou and Yu demonstrated a size approximation protocol for seemingly different model (namely RFID system) that needs $\Omega(\frac{1}{\varepsilon^2 \log \frac{1}{\varepsilon}} + \log \log n)$ slots for $\varepsilon \in [1/\sqrt{n}, 0.5]$ and negligible probability of failure. In fact, this result can be instantly translated into the MAC model.

### 1.3 Our results

We introduce a hierarchy of adaptive adversaries and study their impact on the complexity of performing jobs on a shared channel. The novel and most important parameter of this hierarchy is a partial order restricting adversarial crashes — we call such adversaries ordered. The other parameters are: the number of crashes $f$ (we call them size-bounded adversaries) and a delay $c$ in learning random bits by the adversary (we call them $c$-Round-Delayed or $c$-RD).

We show that adversaries constrained by an order of short width (i.e., with short maximal anti-chain) or 1-RD adversaries have very little power, resulting in performance similar to the one enforced by oblivious adversaries or linearly-ordered adversaries, cf., [13]. More specifically, we develop algorithms ROBAL and GILET, which achieve work performance close to the absolute lower bound $\Omega(t + p\sqrt{t})$ against “narrow-ordered” and 1-RD adversaries, respectively.

In case of ordered adversaries restricted by orders of arbitrary width $k \leq f$, we present algorithm GRUBTECH that guarantees work $O(t + p\sqrt{t} + p \min\{\frac{p}{p-f}, t, k\} \log p)$ against ordered adversaries restricted by orders of width $k$, and show that it is efficient by proving a lower bound for a broad class of partial orders. This also extends the result for a weakly-adaptive linearly-bounded adversary from [13] to any number of crashes $f < p$, as weakly-adaptive adversary is a special case of ordered adversary restricted by a single anti-chain. Our results together with [13] prove separation between classes of adversaries. The easiest to overcome, apart of the oblivious ones, are the following adaptive adversaries: 1-RD adversaries, ordered adversaries restricted by short-width orders, and linearly bounded adversaries. More demanding are ordered adversaries restricted by order of width $k$, for larger values of $k$, and $f$-bounded adversaries for $f$ close to $p$. The most difficult are strongly-adaptive adversaries. See Table 1 for detail results and comparisons.

The hierarchy of the considered adversaries is illustrated on Figure 1. It that depends on three main factors. Additionally, we introduce several solutions for the specified settings. Consequently our contribution is a complement to adversarial scenarios presented in literature together with a taxonomy describing the dependencies between different adversaries.

First of all we have the vertical axis which describes adversary features, that is how restricted his decisions are. We have the Strongly-Adaptive adversary in the origin, who may decide on-line which stations will be crashed. Above is the Weakly-Adaptive adversary who is slightly weaker and has to declare the subset of stations that will be prone to crashes before the algorithm execution. Next we have the Ordered-Adaptive adversary that we introduce in this paper. Apart from declaring the faulty subset, he has to declare the order in which stations will crash.

The horizontal axis describes another particularity, that we introduce in our paper, i.e., the Round-Delay of adversary decisions. Here as well, the configuration is harder in the origin, and a 0-RD adversary is the strongest against which we may execute an algorithm. An interesting particularity is that if
the Strongly-Adaptive adversary becomes delayed by just one round, then we may design a solution that work complexity is independent of the number crashes.

The axis orthogonal to those already considered, describes the channel feedback. In the origin we have a multiple-access channel without collision detection, then there is the beeping channel, followed by MAC with collision detection.

We may see that the most difficult setting is in the origin, while going onwards makes the problem easier. The boxes in Figure 1 represent the algorithms and their work complexities in certain configurations. The bold boxes denote algorithms from this paper and the remaining ones are from CKL [13]. Factors marked red denote the “distance” from the lower bounds, understood as how far the algorithms are from optimum.

| algorithm       | channel | adv. | work                                      | ref.   | lower bound                                           | ref.   |
|-----------------|---------|------|-------------------------------------------|--------|-------------------------------------------------------|--------|
| Two-Lists       | no-CD   | SA   | $O(t + pvT + p \min \{ f, t \})$          | [13] Thm 1 | $O(t + pvT + p \min \{ f, t \})$                        | [13] Thm 2 |
| Groups-Together | CD      | SA   | $O(t + pvT)$                               | [13] Thm 3 | $O(t + pvT)$                                           | [13] Lem 2 |
| Mix-Rand        | no-CD   | WALB | $O(t + pvT)$                               | [13] Thm 5 | $O(t + pvT)$                                           | [13] Lem 2 |
| GrubTech        | no-CD   | WA   | $O(t + pvT + p \min \{ t, k \} \log p)$   | Sec 5  | $O(t + pvT + p \min \{ t, k \} \log p)$               | Sec 6  |
| GrubTech        | no-CD   | COA  | $O(t + pvT + p \min \{ t, k \} \log p)$   | Sec 6  | $O(t + pvT + p \min \{ t, k \} \log p)$               | Sec 6  |
| GILET           | no-CD   | 1-RD | $O(t + pvT \log^2 p)$                     | Sec 7  | $O(t + pvT)$                                           | [13] Lem 2 |
| ROBAL           | no-CD   | LOA  | $O(t + pvT \log p)$                       | Sec 4  | $O(t + pvT)$                                           | [13] Lem 2 |

Table 1: Summary of main results; first three were introduced in CKL [13], the other are presented in this paper. CD stands for collision detection model feature. SA stands for Strongly-Adaptive adversary, WA (WALB) stands for Weakly-Adaptive (Linearly-Bounded) adversary, COA stands for Chain-Ordered adversary, LOA stands for Linearly-Ordered adversary, and 1-RD stands for 1-Round-Delay adversary.

A subset of the results from this paper forms a hierarchy of partially ordered adversaries. The first solution, we introduced was designed to work against a linearly-ordered adversary, whose pattern of crashes is described by a linear order. The upper bound of this algorithm does not depend on the number of crashes and is just logarithmically far from the minimal work complexity in the assumed model. The second algorithm serves for the case when the adversary’s partial order of stations forms a maximum length anti-chain. Nevertheless we also analyze this solution against an in-between situation when the
partial order is shaped by \( k \) chains of an arbitrary length, yet the sum of their lengths is \( f \).

In order to conclude the content of this paper, we would like to emphasize that basing on solutions from CKL [13] we introduce different algorithms and specific adversarial scenarios, for more complex setups what, to some extent, filled the gaps for randomized algorithms solving Do-All in the most challenging adversarial scenarios and communication channels providing least feedback. Due to the basic nature of the considered communication model — a shared channel with acknowledgments only — our solutions are also implementable and efficient in various different types of communication models with contention and failures.

1.4 Our techniques

All our algorithms work on a shared channel with acknowledgments only, without collision detection, what makes the setup challenging. While it was already shown that there is not much we can do against a Strongly-Adaptive \( f \)-Bounded, our goal was to investigate whether there are some other adversaries that an algorithm can play against efficiently.

Taking a closer look at our algorithms, each of them works differently against different adversaries. ROBAL does not simulate collision detection mechanism, opposed to the other two solutions, but tries to exploit good properties of an existing (but a priori unknown to the algorithm) linear order of crashes. On the other hand, its execution against a Weakly-Adaptive Linearly-Bounded could be completely inefficient — the adversary could enforce a significant increase in the overall performance. GRUBECH cannot work efficiently against the 1-RD adversary, as there is a global leader chosen to coordinate the CRASH-ECHO procedure that simulates confirmations in a way similar to collision detection mechanism (recall that we do not assume collision detection given as channel feedback). Hence such an adversary could decide to always crash the leader, making the algorithm futile, as electing a leader is quite costly. Yet from a different angle, GILET confirms every piece of progress by electing a leader in a specific way, which is efficient against 1-RD adversary, but executing it against the Weakly-Adaptive adversary would result in an increase in the overall work complexity.

Different natures of adversaries and properties of algorithms discussed above suggest that it may be difficult to design a single universal solution working efficiently against any of the considered adversaries.

1.5 Document structure

We describe the model of the problem, communication channel details, different adversary scenarios and the complexity measure in Section 2. Section 3 is dedicated to the TWO-LISTS procedure from [13] that is used (sometimes after small modifications) as a toolbox in our solutions. Although this procedure is not essential in our solutions, it is convenient to be used as a sub-procedure, at a lower level, once our algorithms figure out (in a distributed way) which set of stations need to be allocated to which set of tasks (this can be repeated many times in fault-prone execution). In Section 4 we present a randomized algorithm ROBAL solving Do-All in presence of a Linearly-Ordered adversary. In the following Section 5 there is a work-efficient algorithm GRUBECH that simulates a kind of fault-tolerant collision detection on a channel without such feature. This is followed by Section 6 where we adjust this solution for a \( k \)-Chain-Ordered adversary. Finally, we have Section 7 that contains a solution for the 1-RD adversary (algorithm GILET) and Section 8 dedicated to the transition of GROUPS-TOGETHER to the beeping model. We conclude with a short summary in Section 9.

2 The Do-All problem — formal model

The Do-All problem has been introduced by Dwork et al [19] and was considered further in numerous papers [14, 9, 12, 17, 20] under different assumptions regarding the model. In this section we shall formulate the model that we considered, based on that shown in [13].
In general the Do-All problem is modeled as a distributed system of computationally constrained devices, that are expected to perform a number of tasks. We will call those devices processors or simply stations. The main efficiency measure that we use is work, i.e., the total number of processor steps available for computations.

2.1 Stations

In our model we assume having \( p \) stations, with unique identifiers from the set \{1, \ldots, p\}. The distributed system of those devices is synchronized with a global clock, and time is divided into synchronous time slots, called rounds. All the stations start simultaneously at a certain moment. Furthermore every station may halt voluntarily. In this paper by \( n \) we will denote the number of operational, i.e., not crashed, stations.

2.2 Communication

The communication channel for processors is the, widely considered in literature, multiple-access channel \([7, 21]\), where a broadcasted message reaches every operational device. We do not allow simultaneous transmissions of several messages. All our solutions work on a channel without collision detection, hence when more than one message is transmitted at a certain round, then the devices hear a signal unresolvable from the background noise. In our model we assume, that the number of bits possible to broadcast in a single transmission is bounded by \( O(\log p) \), however all our algorithms broadcast merely \( O(1) \) bits, hence we omit the analysis of this complexity.

2.3 Different adversarial scenarios

Processors may fail by crashing, what happens because of an adversary activity. One of the factors that describe the adversary is her power \( f \), that is the total number of failures that may be enforced. We assume that \( 0 \leq f \leq p - 1 \), so always at least one station remains operational until an algorithm terminates. Stations that were crashed neither restart nor contribute to work.

We distinguish the following adversary models:

- **Strongly-Adaptive \( f \)-Bounded**: the only restriction of this adversary is that the total number of failures may not exceed \( f \). In particular all possible failures may happen simultaneously.
- **Weakly-Adaptive \( f \)-Bounded**: the adversary has to declare a subset of \( f \) stations prone to crashes before the algorithm execution.
- **Unbounded**: that is Strongly-Adaptive \((p - 1)\)-Bounded.
- **Linearly-Bounded**: an adversary of power \( f \), where \( f = cp \), for some \( 0 < c < 1 \).

The first four adversary models were already studied in literature \([13, 2, 31]\). We introduce new adversary scenarios, that complement the existing adversary models examined in literature. Their descriptions are to be find below.

2.3.1 The Linearly-Ordered \( f \)-Bounded adversary

Formally, the **Linearly-Ordered \( f \)-Bounded** adversary has to declare a subset of at most \( f \) out of \( p \) stations, that will be prone to crashes. Afterwards the adversary has to choose a permutation \( \pi \), designating the order in which the failures will occur. The adversary may enforce a failure independently from time slots (even \( f \) at the same round), but with respect to the order. This means that station \( \pi(i) \) may be crashed if and only if stations \( \pi(j) \) are already crashed, for all \( j < i \). The notion of the permutation is consistent with a linear partial order.

2.3.2 The \( k \)-Chain-Ordered \( f \)-Bounded adversary

The **\( k \)-Chain-Ordered \( f \)-Bounded** adversary has to declare \( f \) stations that will be prone to crashes where \( 0 \leq f \leq p - 1 \). Additionally, she has to choose a partial order consisting of \( k \) chains of arbitrary length that will
represent in what order these stations may be crashed. In what follows there are \(k\) chains, each of which has length \(f\) at most. We denote \(l_j\) as the length of chain \(j\), and we assume that the sum of lengths of all chains is equal \(f\).

### 2.3.3 The c-RD f-Bounded adversary

The \(c\)-\(RD\) adversary decisions take effect with a \(c\) round delay. This means that if we consider time divided into slots (rounds), then if the adversary decides to interfere with the system (crash a processor) then this will happen after \(c\) rounds. Of course we still consider \(f\)-boundedness of the adversary, but apart from that he may decide online, without declaring which stations will be prone to crashes before the algorithm execution.

A special case of the \(c\)-\(RD\) adversary is a 0-\(RD\) and a 1-\(RD\) adversary model. The definition of the former case is consistent with the Strongly-Adaptive adversary. The latter case may give an answer to the question regarding the matter of how delay influences the difficulty of the problem for a very strong adversary.

### 2.4 Complexity measures

The complexity measure that is mainly used in our analysis is work, as mentioned before. It is the number of available processor steps for computations. This means that each operational station that did not halt contributes a unit of work even if it is idling.

In order to precisely describe this complexity measure let us assume that an execution starts when all the stations begin simultaneously in some fixed round \(r_0\). Let \(r_v\) be the round when station \(v\) halts (or is crashed). Then its work contribution is equal \(r_v - r_0\). Consequently the algorithm complexity is the sum of such expressions over all stations, i.e.:

\[
\sum_{0 \leq v \leq p} (r_v - r_0).
\]

In this paper we also use time and energy complexity measures, which we define as follows.

**Definition 1.** Time complexity of an algorithm execution is the number of steps until all the processors either have halted or have failed.

**Definition 2.** Energy complexity of an algorithm execution is the number of transmissions until all the processors either have halted or have failed.

### 2.5 Tasks and reliability

We expect that processors will perform all \(t\) tasks as a result of executing an algorithm. We assume that tasks are similar (that is each task requires the same number of rounds to be done), independent (they can be performed in any order) and idempotent (every task may be performed many times, even concurrently by different processors).

The similarity of tasks lets us as to consider that one round is sufficient to perform a single task. Furthermore a reliable algorithm satisfies the following conditions in any execution: all the tasks are eventually performed, if at least one station remains non-faulty and each station eventually halts, unless it has crashed.

### 3 Technical preliminaries

In this section we describe a deterministic Two-Lists algorithm from [13] which is used in our solutions as a sub-procedure. It was proved that this algorithm is asymptotically optimal for the Weakly-Adaptive adversary on a channel without collision detection, and its work complexity is \(O(t + p\sqrt{l} + p \min\{f, t\})\).

The characteristic feature of Two-Lists is that its complexity is linear for some setups of \(p\) and \(t\) parameters, describing the number of processors and tasks, respectively (for details, see [5]).
Algorithm 1: EPOCH-TWO-LISTS, code for station \( v \); from [13]

1. set pointer \( \text{Task}_\text{To}_\text{Do}_v \) on list \( \text{TASKS} \) to the initial position of the range \( v \);
2. set pointer \( \text{Transmit} \) to the first item on list \( \text{STATIONS} \);
3. repeat
   // Round 1:
   4. perform the first task on list \( \text{TASKS} \), starting from the one pointed to by \( \text{Task}_\text{To}_\text{Do}_v \), that is in list \( \text{OUTSTANDING}_v \); move the performed task from list \( \text{OUTSTANDING}_v \) to list \( \text{DONE}_v \);
   5. advance pointer \( \text{Task}_\text{To}_\text{Do}_v \) by one position on list \( \text{TASKS} \);
   // Round 2:
   6. if \( \text{Transmit} \) points to \( v \) then
      7. broadcast one bit;
   end
   8. attempt to receive a message;
   // Round 3:
   9. if a broadcast was heard in the preceding round then
      10. for each item \( x \) on list \( \text{DONE}_{\text{Transmit}} \) do
         11. if \( x \) is on list \( \text{OUTSTANDING}_v \) then
             12. move \( x \) from \( \text{OUTSTANDING}_v \) to \( \text{DONE}_v \);
         end
         13. if \( x \) is on list \( \text{TASKS} \) then
             14. remove \( x \) from \( \text{TASKS} \);
         end
      end
   14. if list \( \text{TASKS} \) is empty then
      15. halt;
   end
   16. advance pointer \( \text{Transmit} \) by one position on list \( \text{STATIONS} \);
   end
   17. else
      18. remove the station pointed to by \( \text{Transmit} \) from \( \text{STATIONS} \);
   end
20. until (pointer \( \text{Transmit} \) points to the first entry on list \( \text{STATIONS} \)) or (all tasks in list \( \text{TASKS} \) have been covered in the epoch);

Algorithm 2: Two-Lists, code for station \( v \); from [13]

1. - initialize \( \text{STATIONS} \) to a sorted list of all \( p \) names of stations;
2. - initialize both \( \text{TASKS} \) and \( \text{OUTSTANDING}_v \) to sorted list of all \( t \) names of tasks;
3. - initialize \( \text{DONE}_v \) to an empty list of tasks;
4. - repeat
   5. \( \text{EPOCH-TWO-LISTS} \);
   6. until \( \text{halted} \);
3.1 Basic facts and notation

Two-Lists was designed for a channel without collision detection. That is why simultaneous transmissions were excluded therein. It has been realized by a cyclic schedule of broadcasts (round-robin). This means that stations maintain a transmission schedule and broadcast one by one, accordingly. Because of such design every message transmitted via the channel is legible for all operational stations.

Another important fact about Two-Lists is that stations maintain the list of tasks, what enables them to distinguish which tasks are they responsible for. Both the tasks list and the transmission schedule are maintained as common knowledge. The result of such an approach is that stations may transmit messages of a minimal length, just to confirm that they are still operational and performed their assigned tasks.

Additionally the transmission schedule and tasks list is stored locally on each station, but the way how stations communicate allows to think of those lists as common for all operational stations.

Two-Lists is structured as a loop (see Algorithm 1). Each iteration of the loop is called an epoch. Every epoch begins with a transmission schedule and tasks being assigned to processors. During the execution some tasks are performed and if a station transmits such fact, it is confirmed by removing those certain tasks from list TASKS. However due to adversary activity some stations may be crashed, what is recognized as silence heard on the channel in a round that a station was scheduled to transmit. Stations recognized as crashed are also removed from the transmission schedule. Eventually a new epoch begins with updated lists.

Epochs are also structured as loops (see Algorithm 2). Each iteration is now called a phase, that consists of three consecutive rounds in which station $v$:

1. Performs the first unaccomplished task that was assigned to $v$;
2. $v$ broadcasts one bit, confirming the performance of tasks that were assigned to $v$, if it was $v$’s turn to broadcast. Otherwise $v$ listens to the channel and attempts to receive a message
3. Depending on whether a message was heard $v$ updates its information about stations and tasks.

An epoch consists of a number of phases, that is described by the actual number of operational stations or outstanding tasks. In each epoch there is a repeating pattern of phases that consists of the following three rounds: (1) each operational station performs one task. Next (2) a transmission round takes place, where at most one station broadcasts a message, and the rest of the stations attempt to receive it. The process is ended (3) by an updating round, where stations reconstruct their knowledge about operational stations and outstanding tasks.

3.2 The significance of lists

In the previous section we mentioned the concept of knowledge about stations and tasks, that processors maintain. It was described somehow abstractly, so now we shall explain it in detail. Furthermore we shall provide information on how the stations are scheduled to transmit and how do they know which tasks should they perform.

It is not accidental that the algorithm was named Two-Lists as the most important pieces of information about the system are actually maintained on two lists. The first is list STATIONS. It represents operational (at the beginning of an epoch) processors and sets the order in which stations should transmit in consecutive phases. That list is operated by pointer Transmit, that is incremented after every phase. It points exactly one station in a single iteration, what prevents collisions on the channel. Hence when some station did not broadcast we may recognize that it was crashed and eliminate from STATIONS, setting the pointer to the following device.

The second list is TASKS. It contains outstanding tasks, and the associated pointer is Task_To_Do$_v$, separate for each station. Task assignment is organized in the following way. Let us present processors from list STATIONS as a sequence $\langle v_i \rangle_{1 \leq i \leq n}$, where $n = |\text{STATIONS}|$ is the number of operational stations at the beginning of the epoch. Each station is responsible for some
segment of list TASKS and all segments sum to the whole list. The length of a segment for station $v_i$ equals $i$ in a single epoch. A single task may belong to more than one segment at a time, unless the number of tasks is accordingly greater than the number of stations.

It is noticeable that lists STATIONS and TASKS are treated as common to all the devices, because of maintaining common knowledge. However, in fact every station has a private copy of those lists and operates with appropriate pointers.

Finally, there are additional two lists maintained by each station. The first one is list OUTSTANDING$_v$ and it contains the segment of tasks that station $v$ has assigned to perform in an epoch. The second is list DONE$_v$ and it contains tasks already performed by station $v$. These two additional lists are auxiliary and their main purpose is to structure algorithms in a clear and readable way.

### 3.3 Sparse vs dense epochs

The last important element of Two-Lists description, that explains some subtleties are definitions of dense and sparse epochs.

**Definition 3.** Let $n = |STATIONS|$ denote the number of operational stations at the beginning of the epoch. If $n(n+1)/2 \geq |TASKS|$ then we say that an epoch is dense. Otherwise we say that an epoch is sparse.

The expression $n(n+1)/2 = 1 + 2 + \cdots + n$ from the definition above determines how many tasks may be performed in a single epoch. If all the broadcasts in Two-Lists are successful, then this is the number of performed (and confirmed) tasks.

In general that is why if we consider a dense epoch, then it is possible that some task $i$ was assigned more than once to different stations. A dense epoch may end when the list of tasks will become empty. However for sparse epochs the ending condition is consistent with the fact that every station had a possibility to transmit, and pointer Transmit passed all the devices on list STATIONS.

We shall end this section with results from [13] stating that Two-Lists is asymptotically work optimal, for the channel without collision detection and against the Strongly-Adaptive adversary.

**Fact 1.** ([13], Theorem 1) Algorithm Two-Lists solves Do-All with work $O(t + p\sqrt{t} + p \min\{f, t\})$ against the $f$-Bounded adversary, for any $0 \leq f < p$.

**Fact 2.** ([13], Theorem 2) The $f$-Bounded adversary, for $0 \leq f < p$, can force any reliable, possibly randomized, algorithm for the channel without collision detection to perform work $\Omega(t + p\sqrt{t} + p \min\{f, t\})$.

**Fact 3.** ([13], Corollary 1) Algorithm Two-Lists is optimal in asymptotic work efficiency, among randomized reliable algorithms for the channel without collision detection, against the adaptive adversary who may crash all but one station.
4 ROBAL — Random Order Balanced Allocation Lists

In this section we describe and analyze the algorithm for the Do-All problem in the presence of a Linearly-Ordered adversary on a channel without collision-detection. Its expected work complexity is $O(t + p\sqrt{t}\log(p))$ and it uses the Two-Lists procedure from [13] (c.f., Section 3).

Algorithm 3: MIX-AND-TEST, code for station $v$

| Input: $i, t, p$
| 1 $\text{coin} := \frac{p}{2}$;
| 2 for $\sqrt{t}\log(p)$ times do
| 3 $\text{if } v \text{ has not been moved to front of list STATIONS yet then}$
| 4 $\text{toss a coin with the probability } \text{coin}^{-1} \text{ of heads to come up}$
| 5 $\text{end}$
| 6 $\text{if heads came up in the previous step then}$
| 7 $\text{broadcast } v \text{ via the channel and attempt to receive a message}$
| 8 $\text{end}$
| 9 $\text{if some station } w \text{ was heard then}$
| 10 $\text{move station } w \text{ to the front of list } \text{STATIONS};$
| 11 $\text{decrement } \text{coin} \text{ by 1;}$
| 12 $\text{end}$
| 13 $\text{end}$
| 14 $\text{if at least } \sqrt{t} \text{ broadcasts were heard then}$
| 15 $\text{return true; }$
| 16 $\text{end}$
| 17 else
| 18 $\text{return false; }$
| 19 $\text{end}$

Our algorithm changes the order of stations on list STATIONS. Precisely, stations that performed successful broadcasts are moved to front of that list. This procedure has two purposes. On one hand changing the order makes the adversary less flexible in crashing stations, as his order is already determined. On the other hand, we may predict with high probability to which interval $n \in \left(\frac{p}{2^i}, \frac{p}{2^{i+1}}\right]$ for $i = 1, \cdots, \lceil \log_2(p) \rceil$ does the current number of operational stations belong, what is important from the work analysis perspective.

![Figure 3](image-url)

Figure 3: (1) Initially we have $p$ stations. (2) The adversary chooses $f$ stations prone to crashes. (3) Then he declares the order according to which the stations will crash. (4) MIX-AND-TEST chooses a number of leaders which are expected to be distributed uniformly among the adversary linear order.

Stations moved to front of list STATIONS are called leaders. Leaders are chosen in a random process, so we expect that they will be uniformly distributed in the adversary order between stations that were not chosen as leaders. This allows us to assume that a crash of a leader is likely to be preceded
Algorithm 4: ROBAL, code for station $v$

1. if $p^2 \leq t$ then
2. 1. execute TWO-LISTS;
3. end
4. else
5. 1. initialize STATIONS to a sorted list of all $p$ names of stations;
6. 2. if $\log_2(p) > e^{\frac{\sqrt{t}}{2}}$ then
7. 3. execute a $t$-phase epoch where every station has all tasks assigned (without transmissions);
8. 4. execute CONFIRM-WORK;
9. end
10. else
11. 1. $i = 0$;
12. repeat
13. 1. if $(\frac{p}{2}) \leq \sqrt{t}$ then
14. 2. Execute TWO-LISTS;
15. end
16. 2. if MIX-AND-TEST($i$, $t$, $p$) then
17. 3. repeat
18. 4. Execute $\sqrt{t}$ phases of TWO-LISTS;
19. 5. until less than $\frac{1}{4}\sqrt{t}$ broadcasts are heard;
20. end
21. 6. increment $i$ by 1;
22. until $i = \lceil \log_2(p) \rceil$;
23. end
24. end

Algorithm 5: CONFIRM-WORK, code for station $v$

1. $i := 0$;
2. repeat
3. 1. $\text{coin} := \frac{p}{2}$;
4. toss a coin with the probability $\text{coin}^{-1}$ of heads to come up;
5. if heads came up in the previous step then
6. 1. broadcast $v$ via the channel and attempt to receive a message;
7. end
8. if some station $w$ was heard then
9. 1. clear list TASKS;
10. 2. break;
11. end
12. else
13. 1. increment $i$ by 1;
14. 2. if $i = \lceil \log_2(p) \rceil + 1$ then
15. 3. $i := 0$;
16. end
17. end
18. until a broadcast was heard;
by several crashes of other stations.

Let us consider procedure MIX-AND-TEST. If \( n \) is the previously predicted number of operational stations, then each of the stations tosses a coin with the probability of success equal \( 1/n \). In case where none or more than one of the stations broadcasts then silence is heard on the channel, as there is no collision detection. Otherwise, when only one station did successfully broadcast it is moved to front of list STATIONS and the procedure starts again with a decremented parameter. However stations that have already been moved to front do not take part in the following iterations of the procedure.

Another important feature of our algorithm is that we do not perform full epochs, but \( \sqrt{t} \) phases of a TWO-LISTS epoch. This allows us to be sure that the total work accrued in each epoch does not exceed \( p\sqrt{t} \).

Before the algorithm execution the Linearly-Ordered adversary has to choose \( f \) stations prone to crashes and declare an order that will describe in what order those crashes may happen. In what follows, when there are unsuccessful broadcasts of leaders (crashes) we may be approaching the case when \( n \leq \sqrt{t} \) and we can execute TWO-LISTS that complexity is linear in \( t \) for such parameters. Alternatively the adversary spends the majority of his possible crashes and the stations may finish all the tasks without any distractions.

Overall the design of ROBAL is quite simple: if some specific conditions are not satisfied (see Algorithm 4 lines 1-10), then MIX-AND-TEST is executed, and hence a number of leaders is chosen. They work for \( \sqrt{t} \) phases, and if no crashes occur, they will finish doing all the tasks within \( O(1) \) epochs (because \( \sqrt{t} \) stations perform \( 1 + \cdots + \sqrt{t} = O(t) \) tasks in a single epoch). Otherwise, if there were crashes then we may have to repeat the testing procedure and draw another set of leaders. Alternatively, if \( n < \sqrt{t} \) is satisfied, we may execute the TWO-LISTS algorithm that work complexity is linear in \( t \) for such parameters.

4.1 Analysis of ROBAL

**Lemma 1.** Algorithm ROBAL is reliable.

**Proof.** We need to show that all the tasks will be performed as a result of executing the algorithm. First of all, if we fall in to the case when \( \frac{p}{2t} \leq \sqrt{t} \) (or initially \( p \leq \sqrt{t} \)) then TWO-LISTS is executed, which is reliable as we know from [13].

Secondly, when \( \log_2(p) > e \frac{\sqrt{t}}{2} \) we assign all the tasks to every station and let the stations work for \( t \) phases. We know that \( f < p \) so at least one station will perform all the tasks.

Finally, if those conditions do not hold, the algorithm runs an external loop in which variable \( i \) increments after each iteration. If the loop is performed \( \lceil \log_2(p) \rceil \) times then we run TWO-LISTS. Variable \( i \) may not be incremented only if the algorithm will enter and stay in the internal loop. However this is possible only after performing all the tasks, because the internal loop runs for a constant number of times until all tasks are completed.

**Fact 4.** TWO-LISTS always solves the Do-All problem with \( O(pt) \) work.

\( O(pt) \) work is consistent with a scenario when every station performs every task. Comparing it with how TWO-LISTS works, justifies the fact.

We already mentioned that ROBAL was modeled in such a way, that whenever \( \frac{p}{2t} \leq \sqrt{t} \) holds, the TWO-LISTS algorithm is executed, because its complexity for such parameters is \( O(t) \). We shall prove it in the following.

**Fact 5.** Let \( n \) be the number of operational processors, and \( t \) be the number of outstanding tasks. Then for \( n \leq \sqrt{t} \) TWO-LISTS work complexity is \( O(t) \).

**Proof.** If \( n \leq \sqrt{t} \), then the outstanding number of crashes is \( f < n \), hence \( f < \sqrt{t} \). Algorithm TWO-LISTS has \( O(t + p\sqrt{t} + p \min\{f, t\}) \) work complexity. In what follows the complexity is \( O(t + \sqrt{t}\sqrt{t} + \sqrt{t} \min\{\sqrt{t}, t\}) = O(t) \).
Lemma 2. Let us assume that we have \( n \) operational stations at the beginning of an epoch, where \( \sqrt{t} \) were chosen leaders. If the adversary crashes \( n/2 \) stations, then the probability that there were \( 3/4 \) of the overall number of leaders crashed in this group does not exceed \( e^{-\frac{1}{8}\sqrt{t}} \).

Proof. We have \( n \) stations, among which \( \sqrt{t} \) are leaders. The adversary crashes \( n/2 \) stations and our question is how many leaders were in this group?

The hypergeometric distribution function with parameters \( N \) - number of elements, \( K \) - number of highlighted elements, \( l \) - number of trials, \( k \) - number of successes, is given by:

\[
P[X = k] = \frac{\binom{K}{k} \binom{N-K}{l-k}}{\binom{N}{l}}
\]

The following tail bound from [27] tells us, that for any \( t > 0 \) and \( p = K/N \):

\[
P[X \geq (p + t)l] \leq e^{-2t^2l}.
\]

Identifying this with our process we have that \( K = n/2 \), \( N = n \), \( l = \sqrt{t} \) and consequently \( p = 1/2 \).

\[
\frac{X}{3/4} \leq e^{-\frac{1}{8}\sqrt{t}}.
\]

Lemma 3. Let us assume that the number of operational stations is in \( (\frac{p}{2}, \frac{p}{2} - 1) \) interval. Then procedure MIX-AND-TEST(\( i, t, p \)) will return true with probability \( 1 - e^{-c \sqrt{t \log_2(p)}} \), for some \( 0 < c < 1 \).

Proof.

Claim 1. Let the current number of operational stations be in \( (\frac{p}{2}, x) \). Then the probability of an event that in a single iteration of MIX-AND-TEST exactly one station will broadcast is at least \( \frac{1}{2\sqrt{e}} \) (where the \( \text{coin}^{-1} \) parameter is \( \frac{1}{2} \)).

Proof. Let us consider a scenario where the number of operational stations is in \( (\frac{p}{2}, x) \) for some \( x \). If every station broadcasts with probability of success equal \( 1/x \) then the probability of an event that exactly one station will transmit is \( (1 - \frac{1}{x})^{x-1} \geq 1/e \). Estimating the worst case, when there are \( \frac{p}{2} \) living stations (and the probability of success remains \( 1/x \)) we have that

\[
\frac{1}{2} \left( 1 - \frac{1}{x} \right)^{x-1} \geq \frac{1}{2\sqrt{e}}.
\]

According to Lemma 2, the probability of an event that in a single round of MIX-AND-TEST exactly one stations will be heard is \( \frac{1}{2\sqrt{e}} \).

We assume that \( n \in (\frac{p}{2}, \frac{p}{2} - 1) \). We will show that the algorithm shall confirm appropriate \( i \) with probability \( 1 - e^{-c \sqrt{t \log_2(p)}} \). For this purpose we need \( \sqrt{t} \) transmissions to be heard.

Let \( X \) be a random variable such that \( X = X_1 + \cdots + X_{\sqrt{t \log_2(p)}} \), where \( X_1, \cdots, X_{\sqrt{t \log_2(p)}} \) are Poisson trials and

\[
X_k = \begin{cases} 
1 & \text{if station broadcasted,} \\
0 & \text{otherwise.}
\end{cases}
\]

We know that

\[
\mu = E[X] = E[X_1 + \cdots + X_{\sqrt{t \log_2(p)}}] \geq \frac{\sqrt{t \log_2(p)}}{2\sqrt{e}}.
\]
To estimate the probability that $\sqrt{t}$ transmissions were heard we will use the Chernoff’s inequality.

We want to have that $(1 - \epsilon)p = \sqrt{t}$. Thus

$$\epsilon = \frac{\sqrt{t}}{p} = \frac{\log_2(p) - 2\sqrt{t}}{\log_2(p)}$$

and $0 < \epsilon < 1$ for sufficiently large $p$. Hence

$$\mathbb{P}[X < \sqrt{t}] \leq e^{-\frac{(\log_2(p) - 2\sqrt{t})^2}{2\sqrt{t} \log_2(p)}} = e^{-\epsilon \sqrt{t} \log_2(p)},$$

for some bounded $0 < c < 1$. We conclude that with probability $1 - e^{-c \sqrt{t} \log_2(p)}$ we shall confirm the correct $i$ which describes and estimates the current number of operational stations.

\textbf{Lemma 4.} \textsc{Mix-And-Test}(i, t, p) will not be executed if there are more than $\frac{p}{2\sqrt{t}}$ operational stations, with probability not less than $1 - (\log_2(p))^2 \max\{e^{-\frac{1}{2} \sqrt{t}}, e^{-c \sqrt{t} \log_2(p)}\}$.

\textbf{Proof.} Let $A_i$ denote an event that at the beginning of and execution of the \textsc{Mix-And-Test}(i, t, p) procedure there were no more than $\frac{p}{2\sqrt{t}}$ operational stations.

The basic case then $i = 0$ is trivial, because initially we have $p$ operational stations, thus $\mathbb{P}(A_0) = 1$.

Let us consider an arbitrary $i$. We know that

$$\mathbb{P}(A_i) = \mathbb{P}(A_i|A_{i-1})\mathbb{P}(A_{i-1}) + \mathbb{P}(A_i|A_{i-1}^c)\mathbb{P}(A_{i-1}^c) \geq \mathbb{P}(A_i|A_{i-1})\mathbb{P}(A_{i-1}).$$

Let us estimate $\mathbb{P}(A_i|A_{i-1})$. Conditioned on that event $A_{i-1}$ holds, we know that after executing \textsc{Mix-And-Test}(i−1, t, p) we had $\frac{p}{2\sqrt{t}}$ operational stations. In what follows if we are now considering \textsc{Mix-And-Test}(i, t, p), then we have two options:

1. \textsc{Mix-And-Test}(i − 1, t, p) returned false,
2. \textsc{Mix-And-Test}(i − 1, t, p) returned true.

Let us examine what do these cases mean:

\textbf{Ad 1.} If the procedure returned false then we know from Lemma [3] that with probability $1 - e^{-c \sqrt{t} \log_2(p)}$ there had to be no more than $\frac{p}{2\sqrt{t}}$ operational stations. If that number would be in $\left(\frac{p}{2\sqrt{t}}, \frac{p}{2\sqrt{t}}\right]$ then the probability of returning false would be less than $e^{-c \sqrt{t} \log_2(p)}$.

\textbf{Ad 2.} If the procedure returned true, this means that when executing it with parameters $(i − 1, f, p)$ we had no more than $\frac{p}{2\sqrt{t}}$ operational stations. Then the internal loop of ROBAL was broken, so according to Lemma [2] we conclude that the overall number of operational stations had to reduce by half with probability at least $1 - e^{-\frac{1}{2} \sqrt{t}}$.

Consequently, we deduce that $\mathbb{P}(A_i|A_{i-1}) \geq (1 - (1 - \log_2(p)) \mathbb{P}(A_i) \geq (1 - \log_2(p)) \mathbb{P}(A_i) \geq 1 - \log_2(p) \max\{e^{-\frac{1}{2} \sqrt{t}}, e^{-c \sqrt{t} \log_2(p)}\}).$

We conclude that the probability that the conjunction of events $A_1, \ldots, A_{\log_2(p)}$ will hold is at least

$$\mathbb{P}(A_i) \geq 1 - (\log_2(p))^2 \max\{e^{-\frac{1}{2} \sqrt{t}}, e^{-c \sqrt{t} \log_2(p)}\}.$$
Proof. In the algorithm we are constantly controlling whether condition \( \frac{p}{2^t} > \sqrt{t} \) holds. If not, then we execute \textsc{Two-Lists} which complexity is \( \mathcal{O}(t) \) for such parameters.

If this condition does not hold initially then we check another one i.e. whether \( \log_2(p) > e^{\frac{\sqrt{t}}{2}} \) holds. For such configuration we assign all the tasks to every station. The work accrued during such a procedure is \( \mathcal{O}(pt) \). However when \( \log_2(p) > e^{\frac{\sqrt{t}}{2}} \) then together with the fact that \( e^x < x \) we have that \( \log_2(p) > t \) and consequently the total complexity is \( \mathcal{O}(p \log(p)) \).

Finally, the successful stations, that performed all the task have to confirm this fact. We demand that only one station shall transmit and if this happens, the algorithm terminates. The expected value of a geometric random variable lets us assume that this confirmation will happen in expected number of \( \mathcal{O}(\log(p)) \) rounds, generating \( \mathcal{O}(p \log(p)) \) work.

When none of the conditions mentioned above hold, we proceed to the main part of the algorithm. The testing procedure by \textsc{Mix-And-Test} for each of disjoint cases, where \( n \in (\frac{p}{2^t}, \frac{p}{2^{t-1}}] \) requires a certain amount of work that can be estimated by \( \mathcal{O}(p\sqrt{t} \log(p)) \), as there are \( \sqrt{t} \log_2(p) \) testing phases in each case and at most \( \frac{p}{2^t} \) stations take part in a single testing phase for a certain case.

In the algorithm we run through disjoint cases where \( n \in (\frac{p}{2^t}, \frac{p}{2^{t-1}}] \). From Lemma 2 we know that when some of the leaders were crashed, then a proportional number of all the stations had to be crashed. When leaders are crashed but the number of operational stations still remains in the same interval, then the lowest number of tasks will be confirmed if only the initial segment of stations will transmit. As a result, when half of the leaders were crashed, then the system still confirms \( \frac{p}{2^t} = \Omega(t) \) tasks. This means that even if so many crashes occurred, \( O(1) \) epochs still suffice to do all the tasks. Summing work over all the cases may be estimated as \( \mathcal{O}(p\sqrt{t}) \).

By Lemma we conclude that the expected work complexity is bounded by:

\[
\left( (\log(p))^2 \max\{e^{-\frac{t}{2}} \sqrt{t}, e^{-\phi} \sqrt{t} \log(p) \} \right) \mathcal{O}(pt + p\sqrt{t} \log^2(p)) + \left( 1 - (\log(p))^2 \max\{e^{-\frac{t}{2}} \sqrt{t}, e^{-\phi} \sqrt{t} \log(p) \} \right) \mathcal{O}(p\sqrt{t} \log(p)) = \mathcal{O}(p\sqrt{t} \log(p)),
\]

where the first expression comes from the fact, that if we entered the main loop of the algorithm then we know that we are in a configuration where \( \log_2(p) \leq e^{\frac{\sqrt{t}}{2}} \). Thus we have that

\[
\frac{pt + p\sqrt{t} \log^2(p)}{e^{\frac{\sqrt{t}}{2}}} \leq \frac{pt + pt \log^2(p)}{e^{\frac{\sqrt{t}}{2}}} \leq \frac{p + p \log^2(p)}{e^{\frac{\sqrt{t}}{2}}} \leq p + p \log(p) = \mathcal{O}(p \log(p)),
\]

what ends the proof. \( \square \)

4.2 Remarks on time and energy complexity

In the presented problem model work complexity describes algorithms performance quite precisely, however we would like to state some general ideas about time complexity and energy consumption, understood as the total number of transmissions by stations.

4.2.1 Time complexity

First of all, we must emphasize that time is not the best choice to describe how efficient the algorithms are, because this strongly depends on how the adversary interferes with the system. To illustrate this fact, let us assume that we have a very strong adversary, whose power \( f = p - 1 \). Imagine that in an execution \( \mathcal{E}_1 \) he decides to crash all the stations at the beginning. Then the total time to perform all the tasks on the single remaining station takes \( \mathcal{O}(t) \) time.

Now let us consider execution \( \mathcal{E}_2 \) where the adversary allows all the stations to perform all but one task \( i \) (productive part). After that he concentrates on crashing every station that has task \( i \) in its schedule (failing part). The productive part is completed within \( \mathcal{O}(\sqrt{t}) \) time, and the failing part takes time \( \mathcal{O}(f) \), so altogether execution \( \mathcal{E}_2 \) requires \( \mathcal{O}(\sqrt{t} + f) \) time for all the tasks to be completed.
This somehow gives a view how hard it is to grasp appropriate bounds for the time complexity in the assumed model. Nevertheless we may introduce a general bound resulting from having \(p - f\) non-faulty stations, that eventually have to perform all the tasks. This takes time \(\frac{t}{p-f}\). Apart from that the adversary has \(f\) possible crashes to perform and in consequence, prolong the execution. This together gives a bound \(O(\frac{t}{p-f} + \min\{f, t\})\), considering the case when there are less tasks than the number of possible crashes.

4.2.2 Transmission energy

For TWO-LISTS it is sufficient to state that it requires \(O(t)\) transmissions to confirm performing all the tasks. ROBAL is modeled in such a way, that it considers \(\log(p)\) disjoint cases, depending on the number of operational stations. We constantly control the condition whether \(p > \sqrt{t}\) and run TWO-LISTS epochs for \(\sqrt{t}\) phases. This results in \(O(1)\) epochs sufficing to perform all the tasks within a single case.

Procedure Mix-And-Test is the most energy-demanding part of the algorithm. We run it \(\log(p)\) times at most, and in every iteration it is expected that exactly one station will transmit. Eventually, with high probability we expect to have \(\sqrt{t}\) successful single transmissions in \(\sqrt{t}\log(p)\) trials.

Falling into the case when \(\log_2(p) > e^{\sqrt{32}}\) is costly from work perspective, but while analyzing the number of transmissions we expect that approximately \(\log(p)\) will be enough to terminate.

Finally, we come to a conclusion that ROBAL has \(O(t + \sqrt{t}\log^3(p))\) expected energy complexity, resulting from the total number of broadcasts.

5 GrubTEch — Groups Together with Echo

In this section we present a randomized algorithm designed to reliably perform Do-All in the presence of a Weakly-Adaptive adversary on a shared channel without collision detection. Its expected work complexity is \(O(t + p\sqrt{t} + p \min\{p/(p - f), t\} \log(p))\). Our solution uses the algorithm GROUPS-TOGETHER from [13] and a newly designed Crash-Echo procedure that works as a kind of fault-tolerant replacement of collision detection mechanism (which is not present in the model). In fact, the algorithm presented here is asymptotically only logarithmically far from matching the lower bound shown in [13], which, to some extent, answers the open question stated therein.

Algorithm 6: GrubTEch; code for station \(v\)

|   |   |
|---|---|
| 1 | - initialize STATIONS to a sorted list of all \(p\) stations; |
| 2 | - arrange all \(p\) names of stations into list GROUPS of groups; |
| 3 | - initialize both TASKS and OUTSTANDING\(_v\) to sorted list of all \(t\) names of tasks; |
| 4 | - initialize DONE\(_v\) to an empty list of tasks; |
| 5 | - initialize leader := Elect-Leader and add the leader to each group; |
| 6 | - repeat |
| 7 |   | Epoch-GROUPS-CE; |
| 8 | - until halted; |

The Crash-Echo procedure. In the seminal CKL paper algorithm GROUPS-TOGETHER worked in the exactly same way as TWO-LISTS with the difference, that stations were arranged into groups. All the stations within a certain group had the same tasks assigned and when it came to transmitting they did it simultaneously. This strongly relied on the collision detection mechanism, as they did not necessarily need to know which station transmitted, but they needed to know that there was progress in tasks performance. That is why if a collision was heard and all the stations within the same group were doing the same tasks, we could deduce that those tasks were actually done.

In our model we do not have collision detection, however we designed a mechanism that provides the same feedback without contributing too much work to the algorithm’s complexity. Strictly speaking
Algorithm 7: Procedure EPOCH-GROUPS-CE; code for station $v$

1. set pointer $\text{Task\_To\_Do}_v$ on list $\text{TASKS}$ to the initial position of the range $v$;
2. set pointer $\text{Transmit}$ to the first item on list $\text{GROUPS}$;
3. repeat
   // Round 1:
   4. perform the first task on list $\text{TASKS}$, starting from the one pointed to by $\text{Task\_To\_Do}_v$, that is in list $\text{OUTSTANDING}_v$;
   5. move the performed task from list $\text{OUTSTANDING}_v$ to list $\text{DONE}_v$;
   6. advance pointer $\text{Task\_To\_Do}_v$ by one position on list $\text{TASKS}$;
   // Rounds 2 & 3:
   7. if $\text{Transmit}$ points to $v$ then
      | execute Crash-Echo;
   end
   8. attempt to receive a pair of messages;
   // Round 4:
   9. if $(\text{silent, loud})$ was heard in the preceding round then
      let $w$ be the first station in the group pointed to by $\text{Transmit}$;
      for each item $x$ on list $\text{DONE}_w$ do
         if $x$ is on list $\text{OUTSTANDING}_v$ then
            move $x$ from $\text{OUTSTANDING}_v$ to $\text{DONE}_v$;
         end
         if $x$ is on list $\text{TASKS}$ then
            remove $x$ from $\text{TASKS}$;
         end
      end
      if list $\text{TASKS}$ is empty then
         halt;
      end
      advance pointer $\text{Transmit}$ by one position on list $\text{GROUPS}$;
   end
   10. else
      if $(\text{loud, loud})$ was heard in the preceding round then
         remove the group pointed to by $\text{Transmit}$ from $\text{GROUPS}$;
      end
      else
         remove $\text{leader}$ from all the groups on list $\text{GROUPS}$;
         $\text{leader} := \text{Elect-Leader}$ and add the leader to each group;
      end
   end
36. until pointer $\text{Transmit}$ points to the first entry on list $\text{GROUPS}$;
37. rearrange all stations in the groups of list $\text{GROUPS}$ into a new version of list $\text{GROUPS}$;

Algorithm 8: Procedure Crash-Echo; code for station $v$

// Round 1:
1. broadcast one bit;
// Round 2:
2. if $v = \text{leader}$ then
   3. broadcast one bit;
end
Algorithm 9: Procedure Elect-Leader; code for station $v$

1. $i := 0;
2. \textbf{repeat}
3. \hspace{1em} \text{coin} := p;
4. \hspace{1em} \text{toss a coin with the probability } \text{coin}^{-1} \text{ of heads to come up;}
5. \hspace{1em} \textbf{if} heads came up in the previous step \textbf{then}
6. \hspace{2em} \text{broadcast } v \text{ via the channel and attempt to receive a message;}
7. \hspace{1em} \textbf{end}
8. \hspace{1em} \textbf{if} some station $w$ was heard \textbf{then}
9. \hspace{2em} \text{leader} := w;
10. \hspace{2em} \text{return leader;}
11. \hspace{1em} \textbf{end}
12. \hspace{1em} \textbf{else}
13. \hspace{2em} increment $i$ by 1;
14. \hspace{1em} \textbf{end}
15. \hspace{1em} \textbf{until} $i < p$;
16. set pointer Transmit$_{STATIONS}$ to the first item on list STATIONS;
17. \textbf{repeat}
18. \hspace{1em} \textbf{if} Transmit$_{STATIONS}$ points to $v$ \textbf{then}
19. \hspace{2em} broadcast one bit;
20. \hspace{2em} attempt to receive a message;
21. \hspace{2em} \textbf{if} some station $w$ was heard \textbf{then}
22. \hspace{3em} \text{leader} := w;
23. \hspace{3em} \text{return leader;}
24. \hspace{2em} \textbf{end}
25. \hspace{1em} \textbf{end}
26. \hspace{1em} advance pointer Transmit$_{STATIONS}$ by one position on list STATIONS;
27. \hspace{1em} \textbf{until} a transmission was heard;
we begin with choosing a leader. His work will be of a dual significance. On one hand he will belong to some group and perform tasks regularly. But on the other hand he will also perform additional transmissions in order to indicate whether there was progress when stations transmitted.

When a group of stations is indicated to broadcast the CRASH-ECHO procedure is executed. It consists of two rounds where the whole group transmits together with the leader in the first one and in the second only the leader transmits. We may hear two types of signals:

- **loud** - a legible, single transmission was heard. Exactly one station transmitted.
- **silent** - a signal indistinguishable from the background noise is heard. None or more than one station transmitted.

Let us examine what are the possible pairs (group & leader, leader) of signals heard in such approach:

- **(silent, loud)** - in the latter round the leader is operational, so he must have been operational in the former round. Because silence was heard in the former round this means that there was a successful transmission of at least two stations one of which was the leader. This is a fully successful case.
- **(loud, loud)** - the former and the latter round were loud, so we conclude that it was the leader who transmitted in both rounds. If the leader belonged to the the group scheduled to transmit, then we have progress; otherwise not.
- **(silent, silent)** - if both rounds were silent we cannot be sure was there any progress. Additionally we need to elect a new leader.
- **(loud, silent)** - when the former round was loud we cannot be sure whether the tasks were performed; a new leader needs to be chosen.

Nevertheless, the Weakly-Adaptive adversary has to declare some $f$ stations that are prone to crashes. The elected leader might belong to that subset and be crashed at some time. When this is examined, the algorithm has to switch to the ELECT-LEADER mode, in order to select another leader. Consequently the most significant question from the point of view of the algorithm’s analysis is what is the expected number of trials to choose a non-faulty leader.

**Two modes.** We need to select a leader and be sure that he is operational in order to have our progress indicator working instead of the collision detection mechanism. When the leader is operational we simply run GROUPS-TOGETHER algorithm with the difference that instead of a simultaneous transmission by all the stations within a group, we run the CRASH-ECHO procedure that allows us to distinguish whether there was progress.

Choosing the leader is performed by procedure ELECT-LEADER, where each station tosses a coin with the probability of success equal $1/p$. If a station is successful then it transmits in the following round. If exactly one station transmits then the leader is chosen. Otherwise the experiment is continued (for $p$ rounds in total). Nevertheless if this still does not work, then the first station that transmits in a round-robin fashion procedure, becomes the leader.

**Groups-Together usage.** We explained already what are the main features of our solution, however we need to emphasize that it is based on algorithms from [13]. This means that the core of the algorithm remains the same, but there are some improvements appropriate for the assumed model.

First of all, in the original GROUPS-TOGETHER algorithm, groups of stations made simple simultaneous transmissions. We replaced it with a two-round CRASH-ECHO procedure, that gives similar feedback as if there was a channel with collision detection.

In order to successfully execute CRASH-ECHO we need a leader. This is chosen at the beginning of the algorithm and assigned as an additional station to every group. Nevertheless tasks are scheduled before the leader is chosen, so primarily he performs tasks that belong to his original group, but apart from that, transmits with other groups.

Furthermore the algorithm responds accordingly to the feedback from CRASH-ECHO procedure. This means that if a leader was recognized as crashed, then a new one is elected, or a group of stations is removed from list GROUPS.

Finally we have a leader election procedure that allows us to choose this special station that will serve as a crash and progress indicator.
5.1 Analysis of GrubTEch

Let us begin the analysis of GRUBTECH by recalling some important results from [13].

Lemma 5. ([13], Lemma 4) Algorithm GROUPS-TOGETHER is reliable.

Theorem 2. ([13], Theorem 3) Algorithm GROUPS-TOGETHER solves Do-All with the minimal work $O(t + p\sqrt{t})$ against the $f$-Bounded adversary, for any $f$ such that $0 \leq f < p$.

Theorem 3. ([13], Theorem 6) The Weakly-Adaptive $f$-Bounded adversary can force any reliable randomized algorithm solving Do-All in the channel without collision detection to perform the expected work $\Omega(t + p\sqrt{t} + p \min \{f/(p-f), t\})$.

In fact the theorem above in [13] stated that the lower bound was $\Omega(t + p\sqrt{t} + p \min \{f/(p-f), t\})$, however the proof relied on the round in which the first successful transmission took place. Hence as it must be at least round number 1 we correct it as follows: $\frac{1}{p-f} + 1 = \frac{1}{p-f} + \frac{f}{p-f} = \frac{p}{p-f}$.

Lemma 6. GRUBTECH is reliable.

Proof. The reliability of GRUBTECH is a consequence of the reliability of Groups-Together. We do not make any changes in the core of the algorithm. Crash-Echo does not affect the algorithm, as it always finishes. Elect-Leader procedure always finishes as well. The first loop is executed for at most $p$ times and then it ends. The second loop awaits to hear a broadcast in a round-robin manner. But we know that $0 \leq f \leq p - 1$, so always one processor remains operational and it will respond.

Lemma 7. Procedure ELECT-LEADER will elect a non-faulty leader in $\log(p) \frac{4p}{p-f}$ trials with probability $1 - \frac{1}{p}$.

Proof. We have $p$ stations from which $f$ are prone to crashes. Hence we have $p-f$ non-faulty stations. That is why the probability of a non-faulty one will respond in the election procedure is at least $(p-f)/p$. We may observe that this probability will increase if we failed in previous executions. In fact, after $f$ executions we may be sure to choose a non-faulty leader. However we shall estimate the probability of our process by an event of awaiting the first success in a number of trials, as our process is stochastically dominated by such a geometric distribution process.

We have a channel without collision detection, so exactly one station has to transmit in order to elect a leader. Let $x$ be the actual number of operational stations. The probability $s$ of the event that a non-faulty station will be elected in the procedure may be estimated as follows:

$$\frac{p-f}{p} \left(1 - \frac{1}{p}\right)^{x-1} \geq \frac{p-f}{p} \left(1 - \frac{1}{p}\right)^{p-1} \geq \frac{p-f}{p} \frac{1}{4} \cdot \frac{p}{p-1} = \frac{p-f}{4p}.$$ 

Let us estimate the probability of awaiting the first success in a number of trials. Let $X \sim Geom((p-f)/4p)$. We know that for a geometric random variable with the probability of success equal $s$:

$$\mathbb{P}(X \geq i) = (1-s)^{i-1}.$$ 

Applying this to our case with $i = \frac{4p}{p-f} \log(p) + 1$ we have that

$$\mathbb{P} \left( X \geq \frac{4p}{p-f} \log(p) + 1 \right) = \left(1 - \frac{1}{\frac{4p}{p-f} \log(p)}\right)^{\frac{4p}{p-f} \log(p)} \leq e^{-\log(p)} = \frac{1}{p}.$$ 

Thus the probability of a complementary event is

$$\mathbb{P} \left( X < \frac{4p}{p-f} \log(p) + 1 \right) > 1 - \frac{1}{p}.$$ 

$\square$
Theorem 4. GRUBTEch solves Do-All in the channel without collision detection with the expected work $O(t + p\sqrt{t} + p \min\{p/(p - f), t\} \log(p))$ against the Weakly-Adaptive $f$-Bounded adversary.

Proof. We may divide the work of GRUBTEch to three components: productive, failing and the one reasoning from electing the leader.

Firstly, the core of our algorithm is the same as GROUPS-TOGETHER with the difference that we have the CRASH-ECHO procedure that takes twice as many transmission rounds. According to Theorem 2, it is sufficient to estimate this kind of work as $O(t + p\sqrt{t})$.

Secondly, there is some work that results from electing the leader. According to Lemma 7, the non-faulty leader will be chosen within $\frac{4p}{p - f} \log(p)$ trials of ELECT-LEADER with high probability. That is why the expected work to elect a non-faulty leader is overall $O(p \frac{p}{p - f} \log(p))$.

Finally, there is some amount of failing work that results from rounds where the CRASH-ECHO procedure indicated that the leader was crashed. However work accrued during such rounds will not exceed the amount of work resulting from electing the leader, hence we state that failing work contributes $O(p \frac{p}{p - f} \log(p))$ as well.

Consequently, we may estimate the expected work of GRUBTEch as

$$\left(1 - \frac{1}{p}\right) O(t + p\sqrt{t} + p \min\{p/(p - f), t\} \log(p)) + \frac{1}{p} O(p^2) = O(t + p\sqrt{t} + p \min\{p/(p - f), t\} \log(p))$$

what ends the proof.

5.2 Time and transmission energy complexity

As mentioned previously time complexity is somehow inadequate for describing the algorithms performance and we may conclude that the same time bound $O(t \frac{1}{p - f} + \min\{f, t\})$ applies for GRUBTEch.

Although, energy consumption is more interesting from the point of view of electing the leader. Precisely, we expect to have chosen a non-faulty leader within $\frac{4p}{p - f} \log(p)$ executions, in each of which exactly one station will respond after a constant number of iterations. This gives us $O(p \frac{p}{p - f} \log(p))$ transmissions.

Apart from that, we have broadcasts performed by groups of stations, but this is consistent with algorithm TWO-LISTS. Stations form groups of size $\sqrt{t}$ and perform simultaneous transmissions, instead of doing it in $\sqrt{t}$ phases as in TWO-LISTS. Broadcasts performed by the leader in CRASH-ECHO procedure does not exceed that number. Consequently, we may estimate the energy consumption resulting from this component as $O(t)$, and this eventually gives $O(t + \min\{p \frac{p}{p - f}, t\} \log(p))$ energetic complexity.

6 How GrubTEch works for other partial orders

The line of investigation originated by ROBAL and GRUBTECH leads to a natural question whether considering some intermediate partial orders of the adversary may provide different work complexities. In this section we answer this question in the positive by examining the GRUBTECH algorithm against the $k$-Chain-Ordered adversary on a channel without collision detection.

6.1 The lower bound

We say that a partial order is a $k$-chain-based partial order if it consists of $k$ disjoint chains such that:

- no two of them have a common successor, and
- the total length of the chains is a constant fraction of all elements in the order.
Theorem 5. For any reliable randomized algorithm solving Do-All on the shared channel and any integer \(0 < k \leq f\), there is a \(k\)-chain-based partial order of \(f\) elements such that the ordered adversary restricted by this order can force the algorithm to perform the expected work \(\Omega(t + p\sqrt{t} + p \min\{f/(p - f), t\})\).

Proof. The part \(\Omega(t + p\sqrt{t})\) follows from the absolute lower bound on reliable algorithms on shared channel. We prove the remaining part of the formula. If \(k > c \cdot f/(p - f)\), for some constant \(0 < c < 1\), then that part is asymptotically dominated by \(p \min\{f/(p - f), t\}\) and it is enough to take the order being an anti-chain of \(f\) elements; clearly it is a \(k\)-chain-based partial order of \(f\) elements, and the adversary restricted by this order is equivalent to the weakly-adaptive adversary, for which the lower bound \(\Omega(p \min\{f/(p - f), t\})\) follows directly from Theorem 3. Therefore, in the reminder of the proof, assume \(k \leq c \cdot f/(p - f)\).

Consider the following strategy of the adversary in the first \(\tau\) rounds, for some value \(\tau\) to be specified later. Each station which wants to broadcast alone in a round is crashed in the beginning of this round, just before its intended transmission. Let \(\mathcal{F}\) be the family of all subsets of stations containing \(k/2\) elements. Let \(\mathcal{M}\) denote the family of all partial orders consisting of \(k\) independent chains of roughly (modulo rounding) \(f/k\) elements each. Consider the first \(\tau = k/2\) rounds. The probability \(\Pr(F)\), for \(F \in \mathcal{F}\), is defined to be equal to the probability of an occurrence of an execution during the experiment, in which exactly the stations with from set \(\mathcal{F}\) are failed by round \(\tau\). Consider an order \(M\) selected uniformly at random from \(\mathcal{M}\). The probability that all elements of set \(F \in \mathcal{F}\) are in \(M\) is a non-zero constant. It follows from the following three observations. First, under our assumption, \(k < f\) (as \(k \leq c \cdot f/(p - f)\) for some \(0 < c < 1\)). Second, from the proof of the lower bound in [13] wrt sets of size \(O(f)\), the probability is a non-zero constant provided in each round we have at most \(c' \cdot f\) crashed processes, for some constant \(0 < c' < 1\). Third, since each successful station can enforce the adversary to fail at most one chain, after each of the first \(\tau = k/2\) rounds there are still at least \(k/2\) chains without any crash, hence at most \(f/2\) crashes have been enforced and the argument from the lower bound in [13] could be applied. To conclude the proof, non-zero probability of not hitting any element not in \(M\) means that there is such \(M \in \mathcal{M}\) that the algorithm does not finish before round \(\tau\) with constant probability, thus imposing expected work \(\Omega(pk)\).

6.2 GrubTEch against the \(k\)-Chain-Ordered adversary

The analysis of GrubTEch against the Weakly-Adaptive adversary relied on electing a leader. Precisely, as we knew that there are \(p - f\) non-faulty stations in an execution, then we expected to elect a non-faulty leader in a certain number of trials.

Nevertheless we could have chosen a faulty station as a leader and the adversary could have chosen to crash that station. However the amount of such failing occurrences would not exceed the number of trials needed to elect the non-faulty one. While considering the \(k\)-Chain-Ordered adversary, these estimates are different.

When a leader is elected then he may belong to the non-faulty set (and this is expected to happen within a certain number of trials) or he may be elected from the faulty set, thus will be placed somewhere in the adversary’s partial order. If the leader was elected in a random process then it will appear in a random part of this order. In what follows we may expect that if the adversary decides to crash the leader, then he will be forced to crash several stations preceding the leader in one of the chains in his partial order. Consequently this is the key reason why the expected work complexity would change against the \(k\)-Chain-Ordered adversary.

Theorem 6. GrubTEch solves Do-All in the channel without collision detection with the expected work \(O(t + p\sqrt{t} + p \min\{p/(p - f), k, t\} \log(p))\) against the \(k\)-Chain-Ordered adversary.

Proof. Because of the same arguments as in Theorem 4 it is expected that a non-faulty leader will be chosen in the expected number of \(O(p \cdot \log(p))\) trials, generating \(O(p \cdot \log(p))\) work.
On the opposite, let us consider what will be the work accrued in phases when the leader is chosen from the faulty set and hence may be crashed by the adversary. According to the adversary’s partial order we have initially \( k \) chains, where chain \( j \) has length \( l_j \). If the leader was chosen from that order then it belongs to one of the chains. We will show that it is expected that the chosen leader will be placed somewhere in the middle of that chain.

Let \( X \) be a random variable such that \( X_j = i \) where \( i \) represents the position of the leader in chain \( j \). We have that 
\[
E[X_j] = \sum_{i=1}^{l_j} i \cdot \frac{1}{l_j} = \frac{(1+l_j)l_j}{2} = \frac{1}{2}l_j.
\]

We can see that if the leader was crashed, this implies that half of the stations forming the chain were also crashed. If at some other points of time, the faulty leaders will also be chosen from the same chain, then by simple induction we may conclude that this chain is expected to be all crashed after \( O(k \log(p)) \) iterations, as a single chain has length \( O(p) \) at most. In what follows if there are \( k \) chains, then after \( O(k \log(p)) \) steps this process will end and we may be sure to choose a leader from the non-faulty subset, because the adversary will spend all his failure possibilities.

Finally, if we have a well serving non-faulty leader then the work accrued is asymptotically the same as in Groups-Together algorithm with the difference that each step is now simulated by the Crash-Echo procedure. This work is equal \( O(t + p\sqrt{t}) \).

Altogether, taking Lemma 7 into consideration, the expected work performance of GrubTEch against the \( k \)-Chain-Ordered adversary is
\[
\left(1 - \frac{1}{p}\right) O(t + p\sqrt{t} + p \min\{p/(p - f), k, t\} \log(p)) + \frac{1}{p} O(p^2)
\]
\[
= O(t + p\sqrt{t} + p \min\{p/(p - f), k, t\} \log(p))
\]
what ends the proof.

6.3 GrubTEch against the adversary limited by arbitrary order

Finally, let us consider the adversary that is limited by arbitrary partial order \( P = (P, \succ) \). We say that two partially ordered elements are incomparable if none of relations \( x \succ y \) and \( y \succ x \) hold. Translating into considered model, this means that the adversary may crash incomparable elements in any sequence during the execution of the algorithm (clearly, only if \( x \) and \( y \) are among \( f \) stations chosen to be crash-prone). By the thickness of a partial order \( P \) we understand the maximal size of an antichain in \( P \).

**Theorem 7.** GrubTEch solves Do-All in the channel without collision detection with the expected work \( O(t + p\sqrt{t} + p \min\{p/(p - f), k, t\} \log(p)) \) against adversary constrained by any order of thickness \( k \).

**Proof.** We assume that the crashes forced by the adversary are constrained by some partial order \( P \). Let us first recall the following lemma.

**Lemma 8.** (Dilworth’s theorem [18]) In a finite partial order, the size of a maximum antichain is equal to the minimum number of chains needed to cover all elements of the partial order.

First let us note that the thickness is the size of maximal antichain in \( P \). Clearly, the adversary choosing some \( f \) stations to be crashed cannot increase the size of the maximal antichain. Thus using Lemma 8 we consider the coverage of the crash-prone stations by at most \( k \) disjoint chains. Finally we fall into the case concluded in Theorem 6 that completes the proof.

7 GILET — Groups with Internal Leader Election Together

In this section we introduce an algorithm for the channel without collision detection that is designed to work efficiently against the 1-RD adversary. Its expected work complexity is \( O(t + p\sqrt{t} \log^2(p)) \). The
algorithm makes use of previously designed solutions from [13], i.e., GROUPS-TOGETHER algorithm, however we implement a major change in how the stations confirm their work (due to the lack of collision detection in the model).

Algorithm 10: GILET; code for station v;

1 - arrange all p names of stations into list GROUPS of groups;
2 - initialize variable k := p/ min{⌈√t⌉, p};
3 - initialize both TASKS and OUTSTANDINGv to sorted list of all t names of tasks;
4 - initialize DONEv to an empty list of tasks;
5 - initialize REMOVEDv to an empty list of stations;
6 - repeat
7    | EPOCH-GROUPS-CW(k);
8 until halted;

In GROUPS-TOGETHER stations are arranged into groups. Assigning stations to groups is as follows. Let n be the smallest number such that n(n + 1)/2 > |TASKS| holds. Stations have their unique identifiers from set {1, . . . , p}. Let gi denote some group i where gi contains the stations that identifiers are congruent modulo i. For this reasons any two groups from GROUPS differ in size by at most 1. Consequently the initial partition results in having min{√t, p} groups.

In our model, there is a channel without collision detection. That is why whenever a group g is scheduled to broadcast, a leader election procedure is executed in order to hear a successful transmission of exactly one station. Because all the stations within g had the same tasks assigned, then if the leader is chosen, we know that the group performed appropriate tasks.

The inherent cost of such an approach of confirming work is that we may not be sure whether removed groups did really crash. The effect is that if all the tasks were not performed and all the stations were found crashed, then we have to execute an additional procedure that will finish performing them reliably.

This is realized by a new list REMOVED containing removed stations, and procedure CHECK-OUTSTANDING which assigns every outstanding task to all the stations. Then if with small probability we have mistakenly removed some operational stations, the algorithm still remains reliable and efficient.

7.1 Analysis of GILET

Lemma 9. GILET is reliable.

Proof. As well as in case of GRUBTECH, the solution does depend on reliability of algorithm GROUPS-TOGETHER, because procedure CONFIRM-WORK always terminates. If we fall into a mistake that some operational station has been removed from list GROUPS, than we execute procedure CHECK-OUTSTANDING that will finish all the outstanding tasks. □

Lemma 10. Assume that the number of operational stations within a group is in (2k/√π, k/√π] interval and the coin parameter is set to k/√π. Then during CONFIRM-WORK a confirming-work broadcast will be performed with probability at least 1 − 1/p.

Proof. We assume that the number of operational stations is in (2k/√π, k/√π]. The probability that exactly one station will broadcast, estimated from the worst case point of view where only 2k/√π stations are operational is 1/2√π, because of the same reason as in the Claim of Lemma 3.

That is why we would like to investigate the first success occurrence in a number of trials with the probability of success equal 1/2√π.
Algorithm 11: Procedure EPOCH-GROUPS-CW; code for station $v$;

Input: $k$

1. set pointer $\text{Task}_v\_\text{To}_\text{Do}_v$ on list $\text{TASKS}_v$ to the initial position of the range $v$;
2. set pointer $\text{Transmit}_v$ to the first item on list $\text{GROUPS}_v$;
3. repeat
   4. perform the first task on list $\text{TASKS}_v$, starting from the one pointed to by $\text{Task}_v\_\text{To}_\text{Do}_v$, that is in list $\text{OUTSTANDING}_v$;
   5. move the performed task from list $\text{OUTSTANDING}_v$ to list $\text{DONE}_v$;
   6. advance pointer $\text{Task}_v\_\text{To}_\text{Do}_v$ by one position on list $\text{TASKS}_v$;
   7. if $\text{Transmit}_v$ points to $v$ then
      8. initialize $i := 0$;
      9. repeat
         10. execute $\text{CONFIRM\_WORK}(k)$;
         11. if a broadcast was heard then
             12. break;
         13. else
             14. increment $i$ by 1;
         15. end
      16. until $i < 4 \log(p)$;
      17. end
   18. if a broadcast was heard in the preceding round then
      19. let $w$ be the first station in the group pointed to by $\text{Transmit}_v$;
      20. for each item $x$ on list $\text{DONE}_w$ do
         21. if $x$ is on list $\text{OUTSTANDING}_v$ then
             22. move $x$ from $\text{OUTSTANDING}_v$ to $\text{DONE}_v$;
         23. end
      24. if $x$ is on list $\text{TASKS}_v$ then
         25. remove $x$ from $\text{TASKS}_v$;
      26. end
      27. end
      28. if list $\text{TASKS}_v$ is empty then
         29. halt;
      30. end
      31. advance pointer $\text{Transmit}_v$ by one position on list $\text{GROUPS}_v$;
      32. end
   33. else
      34. add all the stations from group pointed to by $\text{Transmit}_v$ to list $\text{REMOVED}_v$;
      35. remove the group pointed to by $\text{Transmit}_v$ from list $\text{GROUPS}_v$;
      36. execute $\text{CHECK\_OUTSTANDING}$;
      37. halt;
   38. end
50. until pointer $\text{Transmit}_v$ points to the first entry on list $\text{GROUPS}_v$;
41. rearrange all stations in the groups of list $\text{GROUPS}_v$ into a new version of list $\text{GROUPS}_v$;
Algorithm 12: Procedure \textsc{Confirm-Work}; code for station $v$

\textbf{Input:} $k$

\begin{algorithmic}[1]
\State $j := 0$
\Repeat
\State $\text{coin} := \frac{k}{2j}$
\State toss a coin with the probability $\text{coin}^{-1}$ of heads to come up;
\If {heads came up in the previous step}
\State broadcast $v$ via the channel and attempt to receive a message;
\EndIf
\If {some station $w$ was heard}
\State break;
\EndIf
\State increment $j$ by 1;
\Until {$j < \log(k)$};
\end{algorithmic}

Algorithm 13: Procedure \textsc{Check-Outstanding}; code for station $v$

\begin{algorithmic}[1]
\State - basing on list \textsc{Removed} and list \textsc{Tasks} assign every task to all the processors;
\State - execute a $t$ phase epoch;
\State - clear list \textsc{Tasks};
\end{algorithmic}

Let $X \sim \text{Geom}\left(\frac{1}{2\sqrt{e}}\right)$. We know that for a geometric random variable with the probability of success equal $s$:

$$\mathbb{P}(X \geq i) = (1 - s)^{i-1}.$$ 

Hence we shall apply it for $i = 2\sqrt{e}\log(p) + 1$. We have that

$$\mathbb{P}(X \geq 2\sqrt{e}\log(p) + 1) = \left(1 - \frac{1}{2\sqrt{e}}\right)^{2\sqrt{e}\log(p)} \leq e^{-\log(p)} = \frac{1}{p}.$$ 

Thus

$$\mathbb{P}(X > 2\sqrt{e}\log(p) + 1) > 1 - \frac{1}{p}.$$ 

\[\blacksquare\]

\textbf{Theorem 8.} GI\textsc{Let} performs $O(t + p\sqrt{t}\log^2(p))$ expected work on channel without collision detection against the 1-RD adversary.

\textbf{Proof.} The proof of \textsc{Groups-Together} work performance from \cite{13} stated that noisy sparse epochs contribute $O(t)$ to work and silent sparse epochs contribute $O(p\sqrt{t})$. Dense epochs do also contribute $O(p\sqrt{t})$ work. Let us compare this with our solution.

Noisy sparse epochs contribute $O(t)$ because these are phases with successful broadcasts. And there are clearly $t$ tasks to perform, so at most $t$ transmissions will be necessary for this purpose.

Silent sparse epochs, as well as dense epochs consist of mixed work: effective and failing. In our case, each attempt of transmitting is now simulated by $O(\log^2(p))$ rounds. That is why the amount of work is asymptotically multiplied by this factor. Hence we have work accrued during silent sparse and dense epochs contributing $O(p\sqrt{t}\log^2(p))$.

However according to Lemma \cite{10} with some small probability we could have mistakenly removed a group of stations from list \textsc{Groups} because \textsc{Confirm-Work} was silent. Eventually the list of groups may be empty, and there are still some outstanding tasks. For such case we execute \textsc{Check-Outstanding}, where all the stations have the same outstanding tasks assigned, and do them for $t$
phases (which actually means until they are all done). It is clear that always at least one station remains operational and all the tasks will be performed. Work contributed in such case is $O(pt)$.

Let us now estimate the expected work:

$$\left(1 - \frac{1}{p}\right) O(t + p\sqrt{t \log^2(p)}) + \frac{1}{p} O(pt) = O(t + p\sqrt{t \log^2(p)}),$$

what completes the proof.

8 Transition to the beeping model

To this point we considered a communication model based on a shared channel, with distinction that collision detection is not available. In this section we consider the beeping model that is constantly discussed in some recent articles [5].

In the beeping model we distinguish two types of signals. One is silence, where no station transmits. The other is a beep, which, when heard, indicates that at least one station transmitted.

It differs from the channel with collision detection by providing slightly different feedback, but as we show it has the same complexity with respect to reliable Do-All. More precisely, we show that the feedback provided by the beeping channel allows to execute algorithm GROUPS-TOGETHER from [13] and that it is work optimal as well.

8.1 Lower bound

We state the lower bound for Do-All in the beeping model in the following lemma.

**Lemma 11.** A reliable algorithm, possibly randomized, with the beeping communication model performs work $\Omega(t + p\sqrt{t})$ in an execution in which no failures occur.

**Proof.** The proof is an adaptation of the proof of Lemma 1 from [13] to the beeping model. Let $A$ be a reliable algorithm. The part $\Omega(t)$ of the bound follows from the fact that every task has to be performed at least once in any execution of $A$.

Task $\alpha$ is confirmed at round $i$ of an execution of algorithm $A$, if either a station performs a beep successfully and it has performed $\alpha$ by round $i$, or at least two stations performed a beep simultaneously and all of them have performed task $\alpha$ by round $i$ of the execution. All of the stations broadcasting at round $i$ and confirming $\alpha$ have performed it by then, so at most $i$ tasks can be confirmed at round $i$. Let $E_1$ be an execution of the algorithm when no failures occur. Let station $v$ come to a halt at some round $j$ in $E_1$.

**Claim:** The tasks not confirmed by round $j$ were performed by $v$ itself in $E_1$.

**Proof.** Suppose, to the contrary, that this is not the case, and let $\beta$ be such a task. Consider an execution, say $E_2$, obtained by running the algorithm and crashing any station that performed task $\beta$ in $E_1$ just before it was to perform $\beta$ in $E_1$, and all the remaining stations, except for $v$, crashed at step $j$. The broadcasts on the channel are the same during the first $j$ rounds in $E_1$ and $E_2$. Hence all the stations perform the same tasks in $E_1$ and $E_2$ till round $j$. The definition of $E_2$ is consistent with the power of the Unbounded adversary. The algorithm is not reliable because task $\beta$ is not performed in $E_2$ and station $v$ is operational. This justifies the claim.

We estimate the contribution of the station $v$ to work. The total number of tasks confirmed in $E_1$ is at most

$$1 + 2 + \ldots + j = O(j^2).$$

Suppose some $t'$ tasks have been confirmed by round $j$. The remaining $t - t'$ tasks have been performed by $v$. The work of $v$ is at least

$$\Omega(\sqrt{t'} + (t - t')) = \Omega(\sqrt{t}),$$

which completes the proof.
8.2 How algorithm GROUPS-TOGETHER works in the beeping model

Shared channel with collision detection provides three types of signals:

- **Silence** - no station transmits, and only a background noise is heard;
- **Single** - exactly one station transmits a legible information;
- **Collision** - an illegible signal is heard (yet different from Silence), when more than one station transmits simultaneously.

Collision detection was a significant part of algorithm GROUPS-TOGETHER as it provided the possibility of taking advantage of simultaneous transmissions. Because of maintaining common knowledge about the tasks assigned to groups of stations we were not interested in the content of the transmission but the fact that at least one station from the group remained operational, what guaranteed progress.

In the beeping model we cannot distinguish between Single and Collision, however in the sense of detecting progress the feedback is consistent. It means that if a group $g$ is scheduled to broadcast at some phase $i$, then we have two possibilities. If Silence was heard this means that all the stations in group $g$ were crashed, and their tasks remain outstanding. Otherwise if a beep is heard this means that at least one station in the group remained operational. As the transmission was scheduled in phase $i$ this means that certain $i$ tasks were performed by group $g$.

Lemma 11 together with the work performance of GROUPS-TOGETHER allows us to conclude that the solution is also optimal in the beeping model.

**Corollary 1.** Groups-Together is work optimal in the beeping channel against the $f$-Bounded adversary.

9 Conclusions

We addressed the challenge of performing work on a shared channel with crash-prone stations against ordered adversaries, introduced in this work. The considered model is very basic, therefore our solutions could be implemented and efficient in other related communication models with contention and failures.

We found that some orders of crash events are more costly than the other for the algorithm, in particular, more shallow orders or even slight delays in reading random bits constraining the adversary allow solutions to stay close to absolute lower bound for this problem.

Further study of distributed problems and systems against ordered adversaries seems to be a natural future direction. Another interesting area is to study various extensions of the Do-All setting, such as considering a dynamic model, where additional tasks may appear while algorithm execution, partially ordered sets of tasks, or tasks with different lengths and deadlines. In other words, to develop scheduling theory on a shared channel prone to failures. In all the abovementioned directions, including the one considered in this work, one of the most fundamental questions arises: Is there a universally efficient solution against the whole range of adversarial scenarios?

References

[1] N. Abramson. Development of the alohanet. *IEEE Trans. Inf. Theor.*, 31(2):119–123, September 2006.

[2] Hagit Attiya and Jennifer Welch. *Distributed Computing: Fundamentals, Simulations and Advanced Topics*. John Wiley & Sons, 2004.

[3] Baruch Awerbuch, Andrée W. Richa, and Christian Scheideler. A jamming-resistant MAC protocol for single-hop wireless networks. In Rida A. Bazzi and Boaz Patt-Shamir, editors, *Proceedings of the Twenty-Seventh Annual ACM Symposium on Principles of Distributed Computing, PODC 2008, Toronto, Canada, August 18-21, 2008*, pages 45–54. ACM, 2008.
[4] Michael A. Bender, Jeremy T. Fineman, Seth Gilbert, and Maxwell Young. How to scale exponential backoff: Constant throughput, polylog access attempts, and robustness. In Robert Krauthgamer, editor, Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2016, Arlington, VA, USA, January 10-12, 2016, pages 636–654. SIAM, 2016.

[5] Philipp Brandes, Marcin Kardas, Marek Klonowski, Dominik Pajak, and Roger Wattenhofer. Approximating the size of a radio network in beeping model. In Suomela [42], pages 358–373.

[6] Binbin Chen, Ziling Zhou, and Haifeng Yu. Understanding RFID counting protocols. In Sumi Helal, Ranveer Chandra, and Robin Kravets, editors, The 19th Annual International Conference on Mobile Computing and Networking, MobiCom'13, Miami, FL, USA, September 30 - October 04, 2013, pages 291–302. ACM, 2013.

[7] Bogdan S Chlebus. Randomized communication in radio networks, a chapter. Kluwer Academic Publisher, Drodrecht, 2001.

[8] Bogdan S. Chlebus, Roberto De Prisco, and Alex A. Shvartsman. Performing tasks on synchronous restartable message-passing processors. Distrib. Comput., 14(1):49–64, January 2001.

[9] Bogdan S. Chlebus, Leszek Gasieniec, Dariusz R. Kowalski, and Alexander A. Shvartsman. Bounding work and communication in robust cooperative computation. In Proceedings of the 16th International Conference on Distributed Computing, DISC ’02, pages 295–310, London, UK, UK, 2002. Springer-Verlag.

[10] Bogdan S. Chlebus, Karol Gołąb, and Dariusz R. Kowalski. Broadcasting spanning forests on a multiple access channel. Theory of Computing Systems, 36:711–733, 2003.

[11] Bogdan S. Chlebus and Dariusz R. Kowalski. A better wake-up in radio networks. In Proceedings of the Twenty-third Annual ACM Symposium on Principles of Distributed Computing, PODC ’04, pages 266–274, New York, NY, USA, 2004. ACM.

[12] Bogdan S. Chlebus and Dariusz R. Kowalski. Randomization helps to perform independent tasks reliably. Random Structures and Algorithms, 24(1):11–41, 2004.

[13] Bogdan S. Chlebus, Dariusz R. Kowalski, and Andrzej Lingas. Performing work in broadcast networks. Distributed Computing, 18(6):435–451, 2006.

[14] Bogdan S. Chlebus, Roberto De Prisco, and Alexander A. Shvartsman. Performing tasks on synchronous restartable message-passing processors. Distributed Computing, 14(1):49–64, 2001.

[15] Marek Chrobak, Leszek Gasieniec, and Dariusz Kowalski. The wake-up problem in multi-hop radio networks. In Proceedings of the Fifteenth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA ’04, pages 992–1000, Philadelphia, PA, USA, 2004. Society for Industrial and Applied Mathematics.

[16] Andrea E. F. Clementi, Angelo Monti, and Riccardo Silvestri. Optimal f-reliable protocols for the do-all problem on single-hop wireless networks. In Proceedings of the 13th International Symposium on Algorithms and Computation, ISAAC ’02, pages 320–331, London, UK, UK, 2002. Springer-Verlag.

[17] Roberto De Prisco, Alain Mayer, and Moti Yung. Time-optimal message-efficient work performance in the presence of faults. In Proceedings of the Thirteenth Annual ACM Symposium on Principles of Distributed Computing, PODC ’94, pages 161–172, New York, NY, USA, 1994. ACM.

[18] R. P. Dilworth. A decomposition theorem for partially ordered sets. Annals of Mathematics, 51(1):161–166, 1950.
[19] Cynthia Dwork, Joseph Y. Halpern, and Orli Waarts. Performing work efficiently in the presence of faults. *SIAM J. Comput.*, 27(5):1457–1491, 1998.

[20] Z. Galil, A. Mayer, and Moti Yung. Resolving message complexity of byzantine agreement and beyond. In *Proceedings of the 36th Annual Symposium on Foundations of Computer Science*, FOCS ’95, pages 724–, Washington, DC, USA, 1995. IEEE Computer Society.

[21] Robert G. Gallager. A perspective on multiaccess channels. *IEEE Trans. Information Theory*, 31:124–142, 1985.

[22] Leszek Gaśieniec, Andrzej Pelc, and David Peleg. The wakeup problem in synchronous broadcast systems (extended abstract). In *Proceedings of the Nineteenth Annual ACM Symposium on Principles of Distributed Computing*, PODC ’00, pages 113–121, New York, NY, USA, 2000. ACM.

[23] Chryssis Georgiou, Dariusz R. Kowalski, and Alexander A. Shvartsman. Efficient gossip and robust distributed computation. *Theor. Comput. Sci.*, 347(1-2):130–166, November 2005.

[24] Chryssis Georgiou and Alexander A. Shvartsman. *Cooperative Task-Oriented Computing: Algorithms and Complexity*, volume 2. 2011.

[25] Albert G. Greenberg and Schmuel Winograd. A lower bound on the time needed in the worst case to resolve conflicts deterministically in multiple access channels. *J. ACM*, 32(3):589–596, July 1985.

[26] Maurice Herlihy and Nir Shavit. *The Art of Multiprocessor Programming*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2008.

[27] W. Hoeffding. Probability inequalities for sums of bounded random variables. *Journal American Statistics Association*, 58:13–30.

[28] Piotr Indyk. Explicit constructions of selectors and related combinatorial structures, with applications. In *Proceedings of the Thirteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA ’02, pages 697–704, Philadelphia, PA, USA, 2002. Society for Industrial and Applied Mathematics.

[29] Tomasz Jurdziński, Miroslaw Kutylowski, and Jan Zatopianski. Efficient algorithms for leader election in radio networks. In *Proceedings of the Twenty-first Annual Symposium on Principles of Distributed Computing*, PODC ’02, pages 51–57, New York, NY, USA, 2002. ACM.

[30] Tomasz Jurdziński and Grzegorz Stachowiak. Probabilistic algorithms for the wake-up problem in single-hop radio networks. *Theor. Comp. Sys.*, 38(3):347–367, May 2005.

[31] Paris C. Kanellakis and Alex A. Shvartsman. Efficient parallel algorithms can be made robust. *Distrib. Comput.*, 5(4):201–217, April 1992.

[32] Marek Klonowski, Miroslaw Kutylowski, and Jan Zatopianski. Energy efficient alert in single-hop networks of extremely weak devices. *Theor. Comput. Sci.*, 453:65–74, 2012.

[33] Marek Klonowski and Dominik Pajak. Electing a leader in wireless networks quickly despite jamming. In Guy E. Blelloch and Kunal Agrawal, editors, *Proceedings of the 27th ACM on Symposium on Parallelism in Algorithms and Architectures*, SPAA 2015, Portland, OR, USA, June 13-15, 2015, pages 304–312. ACM, 2015.

[34] J. Komlos and A. Greenberg. An asymptotically fast nonadaptive algorithm for conflict resolution in multiple-access channels. *IEEE Trans. Inf. Theor.*, 31(2):302–306, September 2006.
[35] Dariusz R. Kowalski. On selection problem in radio networks. In *Proceedings of the Twenty-fourth Annual ACM Symposium on Principles of Distributed Computing*, PODC ’05, pages 158–166, New York, NY, USA, 2005. ACM.

[36] Dariusz R. Kowalski and Alex A. Shvartsman. Performing work with asynchronous processors: Message-delay-sensitive bounds. In *Proceedings of the Twenty-second Annual Symposium on Principles of Distributed Computing*, PODC ’03, pages 265–274, New York, NY, USA, 2003. ACM.

[37] Eyal Kushilevitz and Yishay Mansour. An $\omega(d \log(n/d))$ lower bound for broadcast in radio networks. *SIAM J. Comput.*, 27(3):702–712, June 1998.

[38] Charles U. Martel. Maximum finding on a multiple access broadcast network. *Inf. Process. Lett.*, 52(1):7–13, October 1994.

[39] Robert M. Metcalfe and David R. Boggs. Ethernet: Distributed packet switching for local computer networks. *Commun. ACM*, 19(7):395–404, July 1976.

[40] Andréa W. Richa, Christian Scheideler, Stefan Schmid, and Jin Zhang. Competitive and fair medium access despite reactive jamming. In *2011 International Conference on Distributed Computing Systems, ICDCS 2011, Minneapolis, Minnesota, USA, June 20-24, 2011*, pages 507–516. IEEE Computer Society, 2011.

[41] Andréa W. Richa, Christian Scheideler, Stefan Schmid, and Jin Zhang. An efficient and fair MAC protocol robust to reactive interference. *IEEE/ACM Trans. Netw.*, 21(3):760–771, 2013.

[42] Jukka Suomela, editor. *Structural Information and Communication Complexity - 23rd International Colloquium, SIROCCO 2016, Helsinki, Finland, July 19-21, 2016, Revised Selected Papers*, volume 9988 of *Lecture Notes in Computer Science*, 2016.

[43] Dan E Willard. Log-logarithmic selection resolution protocols in a multiple access channel. *SIAM J. Comput.*, 15(2):468–477, May 1986.