Distrib. Comput. (2006) 18(6): 435–451
DOI 10.1007/s00446-005-0153-4

ORIGINAL ARTICLE

Bogdan S. Chlebus · Dariusz R. Kowalski · Andrzej Lingas

Performing work in broadcast networks

Published online: 12 December 2005

Abstract We consider the problem of how to schedule t similar and independent tasks to be performed in a synchronous distributed system of p stations communicating via multiple-access channels. Stations are prone to crashes whose patterns of occurrence are specified by adversarial models. Work, defined as the number of the available processor steps, is the complexity measure. We consider only *reliable* algorithms that perform all the tasks as long as at least one station remains operational. It is shown that every reliable algorithm has to perform work $\Omega(t + p\sqrt{t})$ even when no failures occur. An optimal deterministic algorithm for the channel with collision detection is developed, which performs work $\mathcal{O}(t+p\sqrt{t})$. Another algorithm, for the channel without collision detection, performs work $\mathcal{O}(t + p\sqrt{t} + p \min\{f, t\})$, where $f < p$ is the number of failures. This algorithm is proved to be optimal, provided that the adversary is restricted in failing no more than f stations. Finally, we consider the question if randomization helps against weaker adversaries for the channel without collision detection. A randomized algorithm is developed which performs the expected minimum amount $\mathcal{O}(t + p\sqrt{t})$ of work, provided that the adversary may fail a constant fraction of stations and it has to select failure-prone stations prior to the start of an execution of the algorithm.

The work of the first author is supported by the NSF Grant 0310503. A preliminary version of this paper appeared as "The do-all problem in broadcast networks," in *Proceedings, 20th ACM Symposium on Principles of Distributed Computing*, Newport, Rhode Island, 2001, pp. 117–126.

B. S. Chlebus (✉)
Department of Computer Science and Engineering, University of Colorado at Denver and Health Sciences Center, Denver, CO 80217, USA
E-mail: Bodgan.chlebus@cudenver.edu

D. R. Kowalski
Department of Computer Science, University of Liverpool, Liverpool L69 7ZF, UK

A. Lingas
Department of Computer Science, Lund University, 221 00 Lund, Sweden

Keywords Distributed algorithm · Multiple-access channel · Fail-stop failure · Adversary · Independent tasks

1 Introduction

We consider the problem of how to schedule t similar and independent tasks in a distributed system with p processors prone to failures. This problem is known under the name *Do-All* [1]. The distributed setting we assume in this paper consists of a number of stations communicating via a multiple access channel [2, 3]. The system is synchronous. There is a global clock which defines the same rounds for all stations.

The channel operates according to the following rules: if exactly one station performs a transmission at a round, then the message reaches all the stations, but if at least two stations broadcast simultaneously, then a conflict occurs and no station receives any of these messages. If the stations attached to the channel do not receive a meaningful message at a round, then there are two possible reasons: either none or more than one messages were sent. The ability to distinguish between these two cases is called *collision detection*. When it is available, then the channel is also said to be with *ternary feedback*, because of the following three possible events on the channel recorded by the attached stations: (1) a meaningful message received, (2) no messages sent, and (3) a collision signal received.

Stations are prone to crashes [4, 5]. Allowable scenarios of failure occurrences are determined by adversarial models. An adversary is *size-bounded* if it may fail at most f stations, for a parameter $0 \le f < p$. We may refer to a size-bounded adversary as f*-bounded* to make the value of parameter f explicit. If f is a constant fraction of p, then the adversary is *linearly bounded*. A size-bounded adversary is *weakly adaptive* if it selects, prior to a start of an execution, a subset of stations that might be failed later in the execution at arbitrary times. An f-bounded adversary is *strongly adaptive* if the upper bound f on the number of failures is the only restriction on failure occurrences in an execution.

We want algorithms to perform all the tasks for any pattern of crashes such that at least one station remains operational in an execution. Algorithms that have this property are called *reliable* in this paper. We show that work $\Omega(t + p\sqrt{t})$ has to be performed by a reliable algorithm when no failures occur in an execution. The amount of work asymptotically equal to $t + p\sqrt{t}$ is called *minimal*.

For a channel with collision detection, this lower bound can be attained by a deterministic algorithm against any adversary. The optimal bound on work for a weaker channel without collision detection involves the number of failures f. We give a deterministic algorithm for this channel which performs work $\mathcal{O}(t + p\sqrt{t} + p\,\min\{f, t\})$ against f-bounded adversary. This work complexity is shown to be optimal when the upper bound f on the number of failures is the only restriction on the adversary.

Next we consider the question what is the optimum amount of work needed for the channel without collision detection against adversaries that are weaker than size bounded. We show that a randomized algorithm can achieve the expected minimal work $\mathcal{O}(t + p\sqrt{t})$ against certain weakly-adaptive size-bounded adversaries. The number of faults for this to hold is a constant fraction of the number of all the stations. A conclusion is that randomization helps if collision detection is not available and the adversary is sufficiently restricted. Finally, we show that if the number t of tasks satisfies $t = o(p^2)$, then a weakly-adaptive f-bounded adversary can force any algorithm for the channel without collision detection to perform asymptotically more than the minimal work $\Omega(t + p\sqrt{t})$, provided that $f = p\,(1 - o(1/\sqrt{t}))$.

1.1 Previous work

The *Do-All* problem was introduced by Dwork et al. [1], and investigated in a number of papers [6–10]. All the previous papers considered message-passing models, in which every node can send a message to any subset of nodes in one round. The algorithmic paradigms used included balancing work, checkpointing the progress made, and designating nodes to coordinate by collecting and disseminating information. The primary measures of efficiency of algorithms used in [1] were task-oriented work, in which each performance of a task contributes a unit to complexity, and communication measured as the number of point-to-point messages. That paper also proposed *effort* as a measure of efficiency, which is work and communication combined. The earlier work [1, 6, 9, 10] concentrated on the adversary who could fail all the stations but one. Some recent work [7, 8] concerned optimizing solutions against weaker adversaries.

De Prisco et al. [9] were the first to use the available processor steps [11] as the measure of work for solutions of *Do-All*. They developed an algorithm which has work $\mathcal{O}(t + (f + 1)p)$ and message complexity $\mathcal{O}((f + 1)p)$. Galil et al. [10] improved the message complexity to $\mathcal{O}(fp^{\varepsilon} + \min\{f + 1, \log p\}p)$, for any positive ε, while maintaining the same work complexity. This was achieved as a by-product of their investigation of the Byzantine agreement with stop-failures, for which they found a message-optimal solution. Chlebus et al. [6] studied failure models allowing restarts. Restarted processors could contribute to the task-oriented work, but the cost of integrating them into the system, in terms of the available processor steps and communication, might well surpass the benefits. The solution presented in [6] achieves the work efficiency $\mathcal{O}((t + p\log p + f)\min\{\log p, \log f\})$, and its message complexity is $\mathcal{O}(t + p\log p + fp)$, against suitably defined adversaries that may introduce f failures and restarts. This algorithm is an extension of one that is tolerant of stop-failures and which has work complexity $\mathcal{O}((t + p\log p/\log\log p)\log f)$ and communication complexity $\mathcal{O}(t + p\log p/\log\log p + fp)$. Chlebus and Kowalski [8] 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 $\mathcal{O}(p\log^* p)$, 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 et al. [7] developed a deterministic algorithm with effort $\mathcal{O}(t + p^a)$, 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 complexities $\mathcal{O}(t + p\log^2 p)$ against the strongly-adaptive linearly-bounded adversary. All the previously known deterministic algorithms had either work or communication efficiency $\Omega(t + p^2)$ when as many as a linear fraction of processing units could be failed by the strongly-adaptive adversary. Georgiou et al. [12] developed an algorithm with work $\mathcal{O}(t + p^{1+\varepsilon})$, for any fixed constant ε, by an approach based on gossiping.

Kowalski and Shvartsman [13] 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 $\mathcal{O}((t + pd)\log p)$. Georgiou et al. [14] considered an on-line version of *Do-All*, called *Omni-Do*, in the asynchronous system with partitionable networks. They presented a randomized algorithm achieving a competitive ratio $1 + cw/e$ against an oblivious adversary, where *computational width cw* is a parameter of the poset according to which the adversary splits and merges groups. The same authors studied in [15] an iterative version of *Do-All* modeling a repeated use of *Do-All* solutions. Fernández et al. [16] considered *Do-All* in the model of Byzantine failures.

Clementi et al. [17] investigated *Do-All* in the communication model of a multiple-access channel without collision detection. Their paper [17] appeared shortly after a conference version of this one; 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 [17] 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.

1.2 Related work

The multiple-access channel may be interpreted as a single-hop radio network [2], which is relevant when considering the role of collision detection. Most of the previous research on the multiple-access channel concerned methods of handling packets which the stations keep generating to be broadcast via the channel as quickly as possible; see [2, 3] for surveys of these topics. The packets may be generated in a possibly irregular way, which results in bursty traffic. Techniques like time-division multiplexing are not efficient then, and a better throughput can be achieved by distributing the control among the stations. This is done by randomized conflict-resolution protocols that arbitrate among the stations competing for access to the channel; among the most popular such protocols are Aloha [18] and the exponential backoff [19]. If packets are generated dynamically, then the basic problem is to have stable protocols that prevent the channel from becoming eventually clogged. Recent work in that direction includes the papers of Goldberg et al. [20], Goldberg et al. [21], Håstad et al. [22], and Raghavan and Upfal [23].

Static problems concern a scenario when input data are allocated at the stations prior to the start of an execution. The problem of *selection* 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 [24] developed protocols solving this problem in the expected time $\mathcal{O}(\log\log n)$ in the channel with collision detection. Kushilevitz and Mansour [25] proved a lower bound $\Omega(\log n)$ for the selection problem if collision detection is not available, which shows that there is an exponential gap between the models with collision detection and without it. A related problem of finding maximum among the keys stored in a subset of stations was considered by Martel [26]. There is a related *all-broadcast* problem, in which a subset of some k among all n stations have messages, and all these messages need to be transmitted successfully via the channel as quickly as possible. Komlós and Greenberg [27] showed how to solve this deterministically in time $\mathcal{O}(k + k\log(n/k))$, where both numbers n and k are known. Kowalski [28] gave an explicit solution of complexity $\mathcal{O}(k \text{ polylog } n)$. A lower bound $\Omega(k(\log n)/(\log k))$ was proved by Greenberg and Winograd [29]. Jurdziński et al. [30] studied the related leader election problem for the channel without collision detection. They gave a deterministic solution with sub-logarithmic energy cost and showed a doubly-logarithmic lower bound. Chlebus et al. [31] considered algorithms for broadcasting spanning forests via a multiple-access channel, when the edges of an input graph are stored at the stations.

The *wakeup* problem has as the goal 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 et al. [32] introduced this problem for the multiple-access channel and studied the impact of various modes of synchrony on the complexity of solutions. In particular, they showed that if the stations have access to a global clock, then wakeup can be completed in the expected time $\mathcal{O}(\log n)$ by a randomized protocol. In the case when the local clocks are not synchronized, then there is a randomized protocol working in the expected time $\mathcal{O}(n)$. Jurdziński and Stachowiak [33] improved this result by developing a randomized protocol not relying on a global clock and solving the wakeup problem in the expected time $\mathcal{O}(\log n)$. It was also shown in [32] that deterministic protocols require time $\Omega(n)$, and that there are deterministic protocols working in time $\mathcal{O}(n\log^2 n)$.

The known solutions for the wakeup problem are based on combinatorial structures called radio synchronizers. A binary $n \times m$ array $\mathcal{S}$ is a (n,k)-*synchronizer of length* m if for any non-empty set $A \subseteq \{1,\dots,n\}$ 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. Such synchronizers were defined by Chrobak et al. [34] in the context of their work on the problems of wakeup, leader election and synchronization of local clocks in multi-hop radio networks, but this notion was already implicitly used in [32]. It was shown in [34] that such structures exist with $m = \mathcal{O}(k^2\log n)$, for given n and k. Indyk [35] showed that synchronizers with $k = n$ and $m = \mathcal{O}(n^{1+\varepsilon})$ can be constructed in time $\mathcal{O}(2^{\text{polylog } n})$, for any constant $\varepsilon > 0$. Chlebus and Kowalski [36] proved that (n,k)-synchronizers of length $\mathcal{O}(k^2 \text{ polylog } n)$ can be constructed in time polynomial in n.

1.3 Document structure

We describe the communication environment of a multiple-access channel, the model of failures, and the *Do-All* problem in Sect. 2. This section also includes a discussion of the relevance of the notion of common knowledge to distributed computing on multiple-access channels and shows that any reliable algorithm solving *Do-All* has to perform the minimum amount $\Omega(t + p\sqrt{t})$ of work in an execution without crashes. The next two sections present optimal determinisitic algorithms against strong adversaries that are restricted only by the number of failures they may incur. Section 3 is about the weaker channel, in which collision detection is not available. The next Sect. 4 considers the channel with collision detection. Section 5 addresses the question if randomization can help for the channel without collision detection against weaker adversaries. We conclude with a short discussion in Sect. 6.

2 Distributed environment and problem

In this section we present the distributed environment and the *Do-All* problem. The section includes two technical auxiliary results. One is about common knowledge in distributed computing on multiple-access channels. The other shows that there is a certain minimum amount of work which any algorithm solving *Do-All* in a reliable manner has to perform when no failures occur.

2.1 Processing units

There are p stations, each with a unique identifier in $[p] = \{1, \ldots, p\}$. Number p is known to all the stations. The system is synchronized by a global clock. All stations start simultaneously at round zero with private variables initialized as specified by the code of the executed algorithm. The duration of a global round is related to the time of communication and to the time to process a chunk of input, as explained in detail later. We assume that manipulating private variables takes negligible time. Each station may come to a halt at any time. A station that halted voluntarily is considered to be non-faulty.

2.2 Communication

Stations communicate by broadcasting on a multiple access channel. We assume that all the messages sent via the channel are delivered to all the stations, but if many are sent simultaneously, then they interfere with one another and are received in a garbled form. If a message sent via the channel is successfully delivered to all the stations, in the sense that every recipient can read it correctly, then the message is said to be *heard* by the stations. This happens if exactly one station transmits the message. If no messages are sent, then the stations receive only the *background noise,* which is distinct from any meaningful message. If more than one messages are broadcast simultaneously at a round, then a *conflict* occurs, and no station can hear any of these messages.

We consider two models depending on what feedback stations receive when a conflict for access occurs.

Channel without collision detection: a conflict for access results in each station hearing the background noise.

Channel with collision detection: a conflict for access results in each station hearing the *interference* noise, which is distinct from the background noise.

We often refer to the background noise simply as *silence* and to interference noise as hearing a *collision*. The measure of time is scaled to the time of transmissions; we assume that it takes exactly one round to perform a broadcast via the channel. The size of a packet to carry a single message is assumed to be $\mathcal{O}(\log p)$ bits, but all our deterministic algorithms broadcast messages of merely $\mathcal{O}(1)$ bits.

2.3 Failures

Stations fail by crashing. A station that has not failed by the end of a round is said to be *non-faulty* or *operational* in this round. The rounds in which specific stations crash are controlled by an adversary. Every crash is permanent in the execution, which means that adversarial models allowing restarts, like in [6], are not considered.

Our adversarial models specify restrictions on scenarios to fail stations but do not *require* any minimal pattern nor amount of failures. This means that the weakest adversary does not crash any station.

Efficiency bounds may involve the number of failures f. Next we clarify the role and interpretation of f, which is essential in understanding the meaning of the results. First, f is the number of failures actually occurring in an execution and is never a part of code of protocols. Second, we require all our protocols to be correct even when only one station remains operational in an execution, for whatever price it comes. This turns out to make algorithms costly even when *no failures occur*, as is stated in Lemma 2.

Our upper bounds on complexity with formulas involving f refer to the actual number of failures f in an execution, which is more precise than presenting the bounds as a function of an upper bound on the number of failures. This is relevant to only one algorithm, namely, to the deterministic algorithm for the channel without collision detection, see Theorem 1. The remaining algorithms achieve the minimal complexity that has to be accrued even when no failures occur. This is why their efficiency bounds do not refer to the number of failures f in an execution. Lower bounds also refer to the actual number of failures f in an execution. When this number f is sufficiently small, then it does not affect the asymptotic rate of growth of the expression describing the lower-bound complexity. This phenomenon is again possible because of the strong correctness assumed, see Theorem 2 for an example.

Any result given in this paper is not for an environment determined by the number of failures f only but also for the specific power of some f-bounded adversary. Adversarial models become relevant when we study the impact of randomization in protocols. An adversary is said to be *adaptive with condition $\mathcal{C}$* if decisions about crashes can be made online, the only constraint being that the condition $\mathcal{C}$ has to be satisfied in all executions. We consider the following specific adversaries:

Strongly-Adaptive f-Bounded: the adversary is adaptive with the condition that at most f stations are failed, where $0 \le f < p$.

Weakly-Adaptive f-Bounded: it is adaptive with the following condition:

the adversary needs to select f *failure-prone* stations prior to the start of an execution, and then it may fail only the selected failure-prone stations in the course of this execution,

where $0 \le f < p$.

If algorithms are deterministic, then both the *Strongly-Adaptive f-Bounded* and *Weakly-Adaptive f-Bounded* adversaries are equivalent, because both can determine the behavior of the algorithm in advance. We write simply *f-Bounded* for *Strongly-Adaptive f-Bounded*. The *Unbounded* adversary is the same as *Strongly-Adaptive* $(p-1)$*-Bounded*. The *Linearly-Bounded* adversary denotes *f-Bounded*, where $f = c \times p$, for a constant $0 < c < p$. None of the considered adversaries can fail all the stations in an execution of an algorithm.

2.4 Computation and knowledge

An *execution*, of a given algorithm, is a sequence of configurations of the system in consecutive rounds. The execution is determined by the occurrences of failures of stations, the messages broadcast via the channel, and the sequence of random bits used by each station [37]. If execution $\mathcal{E}'$ is obtained by modifying the actions of an adversary on some other execution $\mathcal{E}$, then it is assumed that each station in $\mathcal{E}'$ receives exactly the same sequence of random bits as in $\mathcal{E}$.

Algorithms are described by referring to variables maintained by stations and by interactions of stations with the channel. Every station has private copies of all variables of the algorithm at hand. If the variable is called X, then the private copy of X used by station v is denoted by X_v. The *state* of a station is determined by the values of all its private variables.

The *knowledge* of a station at a round is determined by the sequence of its previous states and by the history of what has been heard on the channel till this round. Rather than to say that some fact is implied by the knowledge of a station, we say that the station "knows" the fact. All the stations receive the same information from the channel. This property of communication in the system has an impact on what the stations know in common. The notion of *common knowledge*, as defined by Halpern and Moses [38], is relevant to our considerations. It captures a certain natural scope of knowledge which all the stations have together. It does not merely mean what every station knows, but rather what all the stations know that they all know, and on this level only what they know that they know that they know, and so on through the infinity of levels. More precisely, the common knowledge is this portion of knowledge of all stations that belongs to every one of these levels. Clearly, anything successfully broadcast via the channel is common knowledge. Similarly, if an algorithm uses a variable and in all executions the private copies of the variable are guaranteed to have the same value at the end of a round across all the stations, then the current value of this variable at the end of a round is common knowledge. Next we identify what is common knowledge of stations running a distributed deterministic algorithm with communication via a multiple-access channel with possible station crashes.

When station v is operational at a round, then its state is known by all the remaining stations. When station v crashes, then this fact may be common knowledge at some point in the execution. More precisely, at any following round, either all operational stations know that station v has crashed or none of them knows that. Intuitively, this is because there are only few ways to learn that a station crashed, and when this information is provided eventually by the channel, then it is provided to all the stations simultaneously, so it becomes common knowledge. For instance, station v may be scheduled to broadcast as the only station at a round; when the broadcast is not heard, then this justifies a crash of v. Another possibility is that station v is to transmit together with a group of other stations and the broadcast is heard as performed by a different station. The remaining cases require restricting algorithms to have additional properties as defined next.

A deterministic algorithm is *simple* if it has the following two properties:

1. All stations obtain the same input in all executions.
2. If the channel is without collision detection, then whenever at least two stations perform a simultaneous broadcast, then the message sent by each of these stations to the channel allows to retrieve the name of the sender.

Lemma 1 *Consider an execution of a simple deterministic algorithm for the multiple access channel. If a certain station v has crashed by a given round, then either this fact about v is common knowledge or no station knows it at the round. If station v is operational at a round, then the local state of v at this round is common knowledge. Maintaining these two kinds of common knowledge can be implemented, so that referring to it may be made a part of code of algorithms.*

Proof The proof is by induction on the number of rounds. We prove the two parts simultaneously. We also indicate how to maintain the common knowledge for an immediate reference by a station.

The base of induction holds because of two reasons. First, initially all stations may be operational so no station can know about any crashes. Second, the algorithm is simple, hence the individual inputs assigned to the stations are all equal. Therefore the initial state of a station can be determined by the code of the algorithm and the name of the station.

Consider first the knowledge of crashes at a round. Let set A consist of these stations that are both scheduled to perform a transmission in this round and are not known by any operational station to have crashed before. If A is empty, then there is no attempt to transmit and hence no change of knowledge in any station. Suppose A contains exactly one station v. If v is operational, then v broadcasts and all stations hear the message. This does not change the knowledge of all stations about crashes, since all stations know in that respect exactly what v does, by the inductive assumption. Otherwise, if v has crashed, then silence is heard only, and all the stations learn that v has already crashed. Hence all the stations change their knowledge in exactly the same way.

Next consider the case when A contains at least two elements. Suppose first that the channel is without collision detection. If silence is heard, then all the stations learn that either all the elements of A have crashed or at least two of them survived. No station can identify specific failures in A, because even if the station is in A and knows that it is alive, then the behavior of the channel is consistent with no station in A crashing. If a message is heard on the channel, then all the stations receive it. This indicates that exactly one station in A survived till this round while the remaining stations have crashed. The algorithm is simple, hence the message heard on the channel allows for all the stations to get to know the name of the only survivor, since the message contains the name explicitly. Suppose the channel has a collision detection mechanism available. If the collision noise is heard, then every station in set A learns that at least some other station in A is still operational, while all stations not in A learn that at least two stations in A are still operational. This does not identify any new crash, since the collision noise is consistent with no station in A having crashed.

Every station may maintain a list of stations known as having crashed. When this knowledge changes, new stations are simply added to the list. This completes the part of proof about the knowledge of crashes. Next we consider the knowledge of states.

The algorithm is deterministic, so a transition for a given state is uniquely determined. This allows a station to simulate transitions of all the remaining stations in its local memory. The simulating algorithm is structured as follows. Each station has its own private variables, as specified by the executed algorithm, and additionally it maintains copies of private variables of all the remaining stations. A station first modifies its own private variables, according to the code of the executed algorithm. Next the station simulates a similar action, as performed in the current round by all the remaining stations on their private variables, on the copies of private variables of these stations. □

Lemma 1 is proved by describing a simulation that provides common knowledge about states of all stations. All the algorithms presented in this paper are simple. This makes it possible to use the simulation implicitly in pseudo-code of these algorithms, for instance in Fig. 2. The knowledge is sometimes conditional on a simulated station being still operational. Suppose that all stations have some value X obtained as a result of a simulation of the state of station v that is not known as having crashed. Then their common knowledge about station v is in the form of the following implication: "if v is still operational, then the state of v equals X."

2.5 Complexity measure

Work is used as a performance metric. It is defined to be the number of the available-processor steps, which means that each station contributes a unit for each round when it is operational, even when idling, unless it has already crashed or halted. This follows the approach proposed by Kanellakis and Shvartsman [5, 11]. The measure can be defined precisely as follows. We assume that an execution starts when all the stations have been provided with the code to execute and the input, and they know the first round r_0 to begin. For a station v, let r_v be the round when the station halts or crashes, whichever comes first. The contribution of station v to work of the executed algorithm equals the number $r_v - r_0$. The sum $\sum_{1\le v\le p}(r_v - r_0)$ of these contributions over all the stations is the work accrued by the algorithm in the execution.

2.6 Tasks and doing them all

There are t tasks to execute in the system under consideration. The tasks are known to all the stations, in the sense that a station can identify a task in a constant time, given the number of the task. The following three properties of tasks are assumed:

Similar: every task takes the same number of rounds to perform.
Independent: tasks can be performed in any order.
Idempotent: tasks can be performed many times and concurrently.

The *Do-All* problem is defined as follows. Given a collection of t tasks that satisfy the above three properties, a collection of p stations must performs all the tasks in the presence of an adversary who controls failures. We assume for simplicity that it takes exactly one round to perform one task, which does not affect the asymptotic efficiency of algorithms.

The usual approach in the literature is to consider the *Do-All* problem solved when at least one processing unit knows that all the tasks have been performed. The specification of correctness of an algorithm is also normally combined with requiring termination of all processing units and may be restricted to a class of adversaries. We assume a strong notion of correctness of a *Do-All* solution, which requires handling the tasks properly against the strongest size-bounded adversary.

An algorithm solving the *Do-All* problem is *reliable* if the following two conditions are satisfied in any execution:

1. All the tasks are eventually performed, if at least one station remains non-faulty.
2. Each station eventually halts, unless it has crashed.

Observe that when a station halts, in the course of an execution of a reliable algorithm $\mathcal{A}$, then all the tasks have already been performed. This is because otherwise the *Unbounded* adversary could immediately fail the remaining stations and some tasks would remain outstanding forever. This implies that a station that decides to halt *knows* that all the tasks have been already completed in the execution of $\mathcal{A}$. If algorithm $\mathcal{A}$ is both simple and deterministic, then, by Lemma 1, when one station knows that all the tasks have been completed, then all the stations do. It follows that it is possible to simulate $\mathcal{A}$ in such a way that all the stations halt simultaneously. A reliable randomized algorithm is required also to have the property that all the tasks are eventually performed in *all* executions, hence randomization may contribute to efficiency only but not to compromise correctness with some small probability.

2.7 Inherent cost of reliability

Reliability prevents a station to halt when there are still some outstanding tasks. Global coordination of knowledge about the tasks already performed is relatively slow, because, intuitively, it is best achieved by simultaneous transmissions of stations that have been assigned the same set of tasks to perform. Therefore one may expect a certain minimal work to be accrued even in optimistic scenarios with few crashes. This is indeed the case, which can be stated quantitatively as follows:

Lemma 2 *A reliable algorithm, possibly randomized, performs work $\Omega(t + p\sqrt{t})$ in an execution in which no failures occur.*

Proof It is sufficient to consider the channel with collision detection, in which stations could possibly have more information. Let $\mathcal{A}$ 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 $\mathcal{A}$.

Task α is *confirmed* at round i of an execution of algorithm $\mathcal{A}$, if either a station broadcasts successfully and it has performed α by round i, or at least two stations broadcast simultaneously and all of them, with a possible exception of one station, have performed task α by round i of the execution. At least half of the stations broadcasting at round i and confirming α have performed it by then, so at most $2i$ tasks can be confirmed at round i. Let $\mathcal{E}_1$ be an execution of the algorithm when no failures occur. Let station v come to a halt at some round j in $\mathcal{E}_1$.

Claim: The tasks not confirmed by round j were performed by v itself in $\mathcal{E}_1$.

Suppose, to the contrary, that this is not the case, and let β be such a task. Consider an execution, say $\mathcal{E}_2$, obtained by running the algorithm and crashing any station that performed task β in $\mathcal{E}_1$ just before it was to perform β in $\mathcal{E}_1$, and all the remaining stations, except for v, crashed at step j. The broadcasts via the channel are the same during the first j rounds in $\mathcal{E}_1$ and $\mathcal{E}_2$. Hence all the stations perform the same tasks in $\mathcal{E}_1$ and $\mathcal{E}_2$ till round j. The definition of $\mathcal{E}_2$ is consistent with the power of the *Unbounded* adversary. The algorithm is not reliable because task β is not performed in $\mathcal{E}_2$ 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 $\mathcal{E}_1$ is at most

$$2(1 + 2 + \cdots + j) = \mathcal{O}(j^2) .$$

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. □

The amount of work asymptotically equal to $t + p\sqrt{t}$ is called *minimal*. This amount of work is a yardstick that we use to measure efficiency of algorithms in various models of the multiple-access channel. Lemma 2 shows that minimal work is a lower bound on the amount of work performed by a reliable, possibly randomized, algorithm, in the worst case.

Every adversary we consider may choose not to fail any station in an execution, which makes Lemma 2 applicable. Otherwise the proofs of some results, like Corollaries 1 and 2, would not be correct. This is because there are executions in which the amount of performed work is smaller than minimal. For instance, consider the scenario in which $t = o(p^2)$ and all but one stations crash in the beginning. Then the amount of work is $\mathcal{O}(t)$, which is asymptotically smaller than minimal work, because $p = \omega(\sqrt{t})$.

3 Channel without collision detection

In this section we give a work-optimal deterministic solution to the *Do-All* problem for a channel without collision detection. We begin with an overview of the algorithm.

The first decision to make about the algorithm, when pondering its design, is whether to allow for multiple simultaneous transmissions. There is a simple intuition why it is better not to allow conflicts to occur in a setting without conflict detection. Namely, when nothing is heard on the channel, then we cannot distinguish a collision from a situation when all the broadcasting stations have failed. If the algorithm is deterministic, then we anyway would need to make it possible for the participating stations to perform individual broadcasts of their individual progress in performing tasks. We follow this intuition to avoid conflicts between broadcasting stations. There is at most one station scheduled by the algorithm to transmit at a round.

Yet another initial decision regards the schedule and contents of broadcasts. A natural solution would be to have stations broadcast their progress reports in a round-robin way, one station at a time. We apply this in the case when the number of tasks is sufficiently large compared to the number of stations. A message broadcast via the channel consists of just a single bit. Its only purpose is to confirm that the station is still operational. This works because the algorithm is deterministic, hence the states of all operational stations are common knowledge by Lemma 1.

The next decision is about assignment of work to stations. Tasks are maintained as an ordered list. Segments of this list are assigned to stations. Their lengths are increasing to offset delays in scheduled transmissions. A schedule of transmissions and an assignment of tasks need to be coordinated. We partition the sequence of consecutive rounds into segments called epochs. An epoch is a minimal such a segment with the property that either every station has an opportunity to perform one transmission or every task could be reported as performed if there are no crashes. Which of these cases holds depends on the relative magnitude of the

```
– initialize STATIONS to a sorted list of all $p$ names of stations
– initialize both TASKS and $\texttt{OUTSTANDING}_v$ to sorted list of all $t$ names of tasks
– initialize $\texttt{DONE}_v$ to an empty list of tasks
– repeat EPOCH-TWO-LISTS until halted
```

Fig. 1 Algorithm TWO-LISTS; code for station v

```
– set pointer $\texttt{Task\_To\_Do}_v$ on list TASKS to the initial position of the range of $v$
– set pointer Transmit to the first item on list STATIONS
– repeat
   Round 1:– perform the first task on list TASKS, starting from the one pointed to by $\texttt{Task\_To\_Do}_v$, that is in list
             $\texttt{OUTSTANDING}_v$
           – move the performed task from list $\texttt{OUTSTANDING}_v$ to list $\texttt{DONE}_v$
           – advance pointer $\texttt{Task\_To\_Do}_v$ by one position on list TASKS
   Round 2:– if Transmit points to $v$ then broadcast one bit
           – attempt to receive a message
   Round 3: if a broadcast was heard in the preceding round then
           – for each item $x$ on list $\texttt{DONE}_{\texttt{Transmit}}$ do
             – if $x$ is on list $\texttt{OUTSTANDING}_v$ then move $x$ from $\texttt{OUTSTANDING}_v$ to $\texttt{DONE}_v$
             – if $x$ is on list TASKS then remove $x$ from TASKS
           – if list TASKS is empty then halt
           – advance pointer Transmit by one position on list STATIONS
          else remove the station pointed to by Transmit from STATIONS
until (pointer Transmit points to the first entry on list STATIONS) or
          (all tasks in list TASKS have been covered in the epoch)
```

Fig. 2 Procedure EPOCH-TWO-LISTS; code for station v

number of outstanding tasks and the number of operational stations.

The deterministic algorithm we develop in this section is called TWO-LISTS. The algorithm is structured as a repeat loop, see Fig. 1. One iteration of this loop is called an *epoch*. The structure of an epoch is presented in Fig. 2. An explanation how to assign pointer Task_To_Do in the first line in Fig. 2 and when to terminate an epoch is given elsewhere. If, in all executions, all the values of private copies of some variable Y are equal at all stations at the end of every round, and this unique value is common knowledge, then we refer to the private copy Y_v of station v with subscript v dropped. This is the case for the variables STATIONS, Transmit and TASKS in Fig. 2.

An epoch is structured as a repeat loop, see Fig. 2. One iteration of this loop is called a *phase*. A phase takes three rounds containing the following three respective actions:

(1) performing a task by each station; followed by
(2) a single broadcast by a certain station; followed next by
(3) book-keeping operations performed by all stations, which depend on whether the broadcast was heard or not.

One phase involves performing one task by all stations in one round, followed by performing one broadcast by a certain station in one round, so it could be structured to consist of two rounds only. We choose to add one more round for the sake of readability.

Each station maintains private copies of the following four lists: TASKS, STATIONS, DONE, and OUTSTANDING. List TASKS stores all the tasks that have not been announced via the channel as performed yet. List STATIONS contains all stations that either made a broadcast each time they were scheduled to or have not been scheduled to broadcast yet at all. This means that list STATIONS stores stations that could still be operational, because they never failed to broadcast when scheduled to. Lists TASKS and STATIONS are kept sorted, which helps in assigning tasks to perform. Since lists TASKS and STATIONS are defined by what was heard on the channel, their contents are the same in all private copies of stations at the end of a round. It follows that contents of these two lists are global knowledge, which is the reason why the algorithm is called TWO-LISTS. The additional lists are auxiliary and their main purpose is to allow to structure the algorithm in a simple and readable way. When discussing the correctness and efficiency of the algorithm, the size of any used list X is denoted by |X|.

List TASKS shrinks in the course of an execution, until eventually it becomes empty, which indicates that all tasks have been performed; when this happens then all stations halt simultaneously, because the contents of this list is global knowledge. List STATIONS also may shrink, due to failure detected by missing broadcasts, but it always contains at least one item. All input tasks are partitioned into two lists $\texttt{DONE}_v$ and $\texttt{OUTSTANDING}_v$ at node v. The former stores the tasks known by v as already performed and the latter contains the remaining tasks. Every station also maintains copies of these four lists and their corresponding pointers for every other station, for the purpose to simulate the behavior of the whole system.

There is pointer `Transmit` associated with list `STATIONS`. This pointer indicates the station which is scheduled to perform broadcast in the current phase. Pointers `Transmit` have the same value at all stations at the end of a round, which follows by inspection of the code in Figs. 1 and 2. It follows that this value is common knowledge. There is pointer `Task_To_Do` associated with list `TASKS`. This pointer is used to assign the task to perform in the current phase.

3.1 Assigning ranges of tasks to stations

List `TASKS` is considered ordered in the cyclic order. Sequences of items in the list that are consecutive in this order are called *segments*. Each station has such a segment assigned to it at the start of an epoch. This segment is called the *range* of the station for the epoch. The size of the range is called its *length*.

Pointers `Task_To_Do` are assigned their values in the beginning of an epoch as follows. When station y is immediately after station x on list `STATIONS`, then pointer $\texttt{Task\_To\_Do}_y$ is set to the first item in list `TASKS` after the range of tasks assigned to station x. Let list `STATIONS` store the sequence $\langle v_i \rangle_{1\le i\le n}$ of $n = |\texttt{STATIONS}|$ elements. The length of the range of station v_i is set equal to i. A task is said to have been *covered* in the current phase of an epoch, when the task belongs to the union of ranges of all the stations that have broadcast in the epoch by the phase. A task can be assigned to many ranges simultaneously if list `STATIONS` is sufficiently long as compared to the length of list `TASKS` in the beginning of an epoch.

Let list `STATIONS` be of length $n = |\texttt{STATIONS}|$ at the beginning of an epoch. The number of phases in this epoch is at most n. The range of a station consists of the tasks assigned to the station to perform in the epoch by its scheduled transmission. All the ranges for an epoch make together a contiguous interval $\mathcal{I}$ on list `TASKS`, possibly with repetitions of tasks when the assignment of ranges wraps around the list. If station v performs its transmission in the i-th phase of the epoch, then it has performed all i tasks from its range of the interval $\mathcal{I}$ by the transmission. The total number of such tasks in interval $\mathcal{I}$ in the epoch is at most $|\mathcal{I}| = \sum_{i=1}^{n} i = n(n+1)/2$. An epoch is *dense* if at its beginning the inequality $n(n+1)/2 \ge |\texttt{TASKS}|$ holds, otherwise the epoch is called *sparse*. It follows from the definition of sparse and dense epochs, and also from the conditions to terminate an epoch given in Fig. 2, that sparse versus dense epochs may be characterized by the way they terminate. Namely, a sparse epoch terminates when pointers `Transmit` traverse the whole list `STATIONS` and are moved to the first item on this list. A sparse epoch takes n phases. A dense epoch terminates when all the tasks on list `TASKS` have been covered in the epoch, while pointers `Transmit` still have not been moved to the heads of lists `STATIONS`. If there are no failures in a dense epoch, then this epoch suffices to perform all the outstanding tasks and announce this via the channel, because of the inequality $|\mathcal{I}| \ge |\texttt{TASKS}|$ defining dense epochs.

3.2 Updating lists

Lists are updated as follows after a scheduled broadcast. If the broadcast was heard, then the tasks which have been performed by the broadcasting station are removed from `TASKS`, and also moved from lists `OUTSTANDING` to lists `DONE`. If a broadcast was not heard, which indicates a crash, then the station which failed to broadcast is removed from list `STATIONS`. A modification of a pointer for a list is normally explicitly indicated in a pseudo-code. For instance, after a successful broadcast, pointer `Transmit` is advanced by one position on list `STATIONS`. An implicit modification occurs only when the entry pointed to by the pointer is removed from the list. In such a case the pointer is automatically set to the item immediately after the removed one on the list.

Every station v needs to know which tasks have been performed by the station that performed a broadcast at a round, say some station w, because v needs to remove these tasks from its list $\texttt{TASKS}_v$. The pseudo-code in Fig. 2 refers simply to the list $\texttt{DONE}_w$, where $w = \texttt{Transmit}$. This list is private to w but it could be obtained in the following two possible ways. First, it could be obtained from w by a direct broadcast. Station w transmitted anyway so the message broadcast might have contained the contents of list $\texttt{DONE}_w$. We choose to perform broadcasts of constant-size messages, so that they always fit within one round, which excludes this method. Second, local states of stations can be simulated since they are common knowledge by Lemma 1. The approach we use is based on this fact. We have every station simulate the behavior of all the remaining stations in its local memory in a way described in the proof of Lemma 1. The pseudo-code in Fig. 2 does not include these operations, in order to have it simple and readable. The information in Fig. 2 is sufficient, since the operations that a station v performs on the copies of lists of the remaining stations are the same as those given in this pseudo-code that v performs on its own lists.

Lemma 3 *Algorithm* TWO-LISTS *is reliable.*

Proof A copy of list `STATIONS` is initialized to the same sorted list of all stations in every station. The same entry from these lists is removed after a missing broadcast. Hence copies of this list are always the same in all stations. This property, together with the rule to manipulate pointer `Transmit`, assures that both list `STATIONS` and pointer `Transmit` are common knowledge, which guarantees that at most one station is scheduled to broadcast at a round.

Assume that some station, say, v never crashes. Any scheduled broadcast of v is always performed by v alone and so it is always heard. Station v performs at least one new task in each phase, which is guaranteed by the first instruction of Round 1 in Fig. 2. The performed task is next placed on list $\texttt{DONE}_v$. It follows that, when a transmission of station v occurs, list $\texttt{DONE}_v$ is longer than during the preceding broadcast of station v. Therefore all stations remove at least one entry from their copies of list TASKS between two consecutive broadcasts of v.

For any task x, station v eventually either performs x itself or hears about x as performed. When this happens for every task, then list $\texttt{DONE}_v$ contains all the tasks. Now just one broadcast by v results in all lists TASKS becoming empty and all stations halting, unless they halted before. □

The next theorem shows how the efficiency of TWO-LISTS depends on the number of crashes in an execution.

Theorem 1 *Algorithm* TWO-LISTS *solves Do-All with work* $\mathcal{O}(t + p\sqrt{t} + p\min\{f, t\})$ *against the* f*-Bounded adversary, for any* $0 \le f < p$.

Proof We partition the work performed by the stations into three parts: failing, mixed and productive. The *failing* part comprises two kinds of work. First is the work by all stations at a phase in which the scheduled transmission is not heard. Second is the work by a station in the epoch in which the station crashes. The *mixed* work is accrued in certain dense epochs, as will be explained later. The remaining work is called *productive*.

If a station crashes in an epoch, then its work in this epoch equals at most the length of the epoch, which is at most p. Therefore the failing work is $\mathcal{O}(pf)$. Every station performs work $\mathcal{O}(t)$, because the total number of phases is t. It follows that the failing work is also $\mathcal{O}(pt)$. These two facts together give the estimate $\mathcal{O}(p\min\{f, t\})$ on the magnitude of failing work.

The remaining work is estimated separately in dense and sparse epochs. Consider sparse epochs first.

Let $n = |\texttt{STATIONS}|$ be the number of stations assumed still operational at the start of a sparse epoch. Every station, among these n listed in STATIONS, is assigned a range which is a segment of interval $\mathcal{I}$ disjoint from the ranges assigned to other stations. This follows from the inequality $n(n+1)/2 < |\texttt{TASKS}|$ defining sparse epochs. A station, say, v that does not crash in the epoch has its productive work partitioned into a number of parts for the sake of accounting. The first part accounts for the work performed by v by the time of transmission by v in this epoch. This part is assigned to v itself. The next parts are assigned to these stations scheduled to broadcast after v in this epoch that survive past their scheduled times of transmissions. Suppose v is scheduled to transmit during the i-th phase in the current epoch. When the broadcast occurs, then station v has already performed work $3i$ in the current epoch. If we additionally add the work performed in this i-th phase by all the stations that already performed their transmissions in this epoch, then this together makes the amount of work at most $3i + 3(i-1) = 6i - 3$. Hence we can assign up to 6 units of work to every task performed by a station broadcasting in a sparse epoch. This work is categorized as productive. It follows that productive work accrued during sparse epochs is $\mathcal{O}(t)$ in total.

Consider next the work performed during dense epochs. We partition such epochs into two categories. A *mixed* epoch has the property that the number of stations failing to broadcast in their scheduled rounds is more than half of the number of all the phases of the epoch. All the work in a mixed epoch that is not failing is categorized as *mixed*. Observe that the amount of mixed work in a dense epoch is not more than the amount of failing work in the epoch. It follows that the total work in all mixed dense epochs is $\mathcal{O}(p\min\{f, t\})$. A *significant* dense epoch has the property that at least half of the stations scheduled to broadcast in the epoch performed their transmissions. The work accrued during such dense epochs is counted as productive, unless it is counted as failing. Next we estimate such work.

Let t_i be the length of list TASKS at the beginning of the i-th significant dense epoch. Then the duration of this epoch is the smallest positive integer n_i with the property that the inequality

$$n_i(n_i + 1) \ge 2t_i \tag{1}$$

holds. Denote $q_i = \lfloor (n_i - 1)/2 \rfloor$. There are at least q_i successful transmissions in the i-th significant epoch. Observe that the later a failure of a transmission occurs, the more tasks performed by a station fail to be announced via the channel. Therefore the number r_i of new tasks announced as performed in the i-th significant dense epoch is at least

$$\sum_{k=1}^{q_i} k = \frac{q_i(q_i+1)}{2} \ge \left(n_i^2 - 1\right)/4.$$

Use the inequality

$$x^2 - 1 \ge x(x+1)/2\,,$$

which holds for any integer $x \ge 2$. It follows that the inequalities

$$r_i \ge n_i(n_i+1)/8 \ge t_i/4$$

hold, by the estimate (1) for the dense epoch.

We obtain a recursive estimate $t_{i+1} \le 3t_i/4$ on the rate of decreasing of the terms of sequence $\langle t_i \rangle_{i\ge 1}$. To have a bound for the first epoch it is sufficient to take the estimate $t_1 \le t$. It follows that the inequality

$$t_i \le \left(\frac{3}{4}\right)^{i-1} t \tag{2}$$

holds, for $i \ge 1$. There is some constant $c > 0$ such that the contribution to productive work of the i-th significant dense epoch is at most $cp\sqrt{t_i}$. This is because the duration of the epoch is $\mathcal{O}(\sqrt{t_i})$. Combine the estimate $cp\sqrt{t_i}$ with the geometric progression of (2) to obtain that all the significant epochs contribute at most

$$\sum_{i\geq 1} cp\sqrt{t_i} \leq cp\sum_{i\geq 1}\sqrt{t}\left(\frac{\sqrt{3}}{2}\right)^{i-1} = \mathcal{O}(p\sqrt{t})$$

to the productive work.

To summarize, we have shown the following estimates on work. The amounts of failing and mixed work are $\mathcal{O}(p\min\{f,t\})$ each. The amount of productive work is $\mathcal{O}(t)$ in sparse epochs, and it is $\mathcal{O}(p\sqrt{t})$ in dense epochs. □

Next we show a matching lower bound.

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*

$$\Omega(t + p\sqrt{t} + p\ \min\{f,t\})\ .$$

Proof We consider two cases, depending on which term dominates the bound. If it is $\Omega(t + p\sqrt{t})$, then the bound follows from Lemma 2. Consider the case when $\Omega(p \cdot \min\{f,t\})$ determines the magnitude of the bound. Denote $g = \min\{f,t\}$.

Let $\mathcal{E}_1$ be the execution obtained by running the algorithm and crashing any station that wants to broadcast as a single one during the first $g/4$ rounds. Denote as A the set of stations failed in $\mathcal{E}_1$. The definition of $\mathcal{E}_1$ is consistent with the power of the f-*Bounded* adversary, since $|A| \leq g/4 \leq f$.

Claim: No station halts by round $g/4$ in execution $\mathcal{E}_1$.

Suppose, to the contrary, that some station v halts before round $g/4$ in $\mathcal{E}_1$. We show that the algorithm is not reliable. To this end we consider another execution, say, $\mathcal{E}_2$ that may be made to happen by the *Unbounded* adversary. Let γ be a task which is performed in $\mathcal{E}_1$ by at most

$$\frac{pg}{4(t-g/4)} \leq \frac{pg}{3t}$$

stations, except for station v, during the first $g/4$ rounds. It exists because $g \leq t$. Let B be this set of stations. The size $|B|$ of set B satisfies the inequality

$$|B| \leq \frac{pg}{3t} \leq \frac{p}{3}\ .$$

We define operationally a set of stations, denoted C, as follows. Initially C equals $A \cup B$. Notice that the inequality $|A \cup B| \leq 7p/12$ holds. If there is any station that wants to broadcast during the first $g/4$ rounds in $\mathcal{E}_1$ as the only station not in the current C, then it is added to C. At most one station is added to C for each among the first $g/4 \leq p/4$ rounds of $\mathcal{E}_1$, so $|C| \leq 10p/12 < p$.

Let execution $\mathcal{E}_2$ be obtained by failing all the stations in C at the start and then running the algorithm. The definition of $\mathcal{E}_2$ is consistent with the power of the *Unbounded* adversary. There is no broadcast heard in $\mathcal{E}_2$ during the first $g/4$ rounds. Therefore each station operational in $\mathcal{E}_2$ behaves in exactly the same way in both $\mathcal{E}_1$ and $\mathcal{E}_2$ during the first $g/4$ rounds. Task γ is not performed in execution $\mathcal{E}_2$ by round $g/4$, because the stations in B have been failed and the remaining ones behave as in $\mathcal{E}_1$.

The station v is not failed in $\mathcal{E}_2$ and so it performs the same actions in both $\mathcal{E}_1$ and $\mathcal{E}_2$. Consider a new execution, denoted $\mathcal{E}_3$. This execution is like $\mathcal{E}_2$ till round $g/4$, then all the stations, except for v, are failed. The definition of $\mathcal{E}_3$ is consistent with the power of the *Unbounded* adversary. Station v is operational but halted and task γ is still outstanding in $\mathcal{E}_3$ at round $g/4$. We conclude that the algorithm is not reliable. This contradiction completes the proof of the claim.

Let us consider the original execution $\mathcal{E}_1$ again. It follows from the claim that there are at least $p - g/4 = \Omega(p)$ stations still operational and non-halted in round $g/4$ in execution $\mathcal{E}_1$, and they have generated work $\Omega(pg) = \Omega(p \cdot \min\{f,t\})$ by this round. □

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.*

Proof Combine Theorem 1 with Theorem 2. Both hold for the adversary who is adaptive and may crash up to $p - 1$ stations. □

We show in Sect. 5 that randomization can make a difference for the channel without collision detection in weaker adversarial models.

4 Channel with collision detection

In this section we present a deterministic algorithm Groups-Together for the model with collision detection. The algorithm is a modification of algorithm Two-Lists. It is structured similarly as a repeat loop, see Fig. 3. One iteration of the loop is again called an epoch; it is given as procedure Epoch-Groups in Fig. 4. An epoch is structured as a repeat loop, one iteration of the loop is again called a phase. A phase consists of three rounds, of which one is for transmissions. If no transmission occurs in a phase, then the phase is called *silent*, otherwise it is called *noisy*. The mechanism of collision detection allows to distinguish silent phases from those in which multiple transmissions are performed.

The availability of collision detection makes the structure of the algorithm presented in this section differ in two important aspects from algorithm Two-Lists. One is the approach to possible collisions resulting from simultaneous transmissions. Algorithm Groups-Together uses simultaneous transmissions aggressively as a hedge against faults, while Two-Lists avoids any conflicts for access to the channel by its design. The other difference is in balancing the frequency of transmissions of individual stations.

```
– arrange all $p$ names of stations into list GROUPS of groups
– initialize both TASKS and OUTSTANDING$_v$ to a sorted list of all $t$ names of tasks
– initialize DONE$_v$ to an empty list of tasks
– repeat EPOCH-GROUPS until halted
```

Fig. 3 Algorithm GROUPS-TOGETHER; code for station v

```
– set pointer Task_To_Do$_v$ on list TASKS to the initial position of the range of $v$
– set pointer Transmit to the first item on list GROUPS
– repeat
  Round 1:– perform the first task on list TASKS, starting from the one pointed to by Task_To_Do$_v$, that is in list
            OUTSTANDING$_v$
          – move the performed task from list OUTSTANDING$_v$ to list DONE$_v$
          – advance pointer Task_To_Do$_v$ by one position on list TASKS
  Round 2:– if Transmit points to $v$ then broadcast one bit
          – attempt to receive a message
  Round 3: if a broadcast or collision was heard in the preceding round then
          – let $w$ be the first station in the group pointed to by Transmit;
            for each item $x$ on list DONE$_w$, do
             – if $x$ is on list OUTSTANDING$_v$ then move $x$ from OUTSTANDING$_v$ to DONE$_v$
             – if $x$ is on list TASKS then remove $x$ from TASKS
          – if list TASKS is empty then halt
          – advance pointer Transmit by one position on list GROUPS
           else remove the group pointed to by Transmit from list GROUPS
  until pointer Transmit points to the first entry on list GROUPS;
– rearrange all stations in the groups of list GROUPS into a new version of list GROUPS
```

Fig. 4 Procedure EPOCH-GROUPS; code for station v

Algorithm TWO-LISTS allowed for some stations not to make any transmissions during the whole epoch, when the epoch was dense. Algorithm GROUPS-TOGETHER makes every station transmit exactly once during every epoch, including dense epochs.

The different approach to dense epochs results from the availability of collision detection. The algorithm schedules multiple simultaneous transmissions by groups of nodes in such a way that when the last group transmits, all the tasks considered as outstanding in the beginning of the epoch have been covered in the epoch. When at least one station in a group survives till the round when the group is scheduled to transmit, then either a collision or a successful broadcast are heard, which means that the phase is noisy. Collision detection is used to announce progress when many stations perform the same tasks and transmit simultaneously. On the other hand, when all the stations in a group crash by the round when they are scheduled to transmit, then silence on the channel allows to efficiently detect all these multiple crashes in one round, because silence is distinct from collision.

Every station maintains four lists. Lists TASKS, DONE and OUTSTANDING are the same as in Sect. 3. List STATIONS is replaced by list GROUPS which stores groups of stations. List GROUPS is rearranged in the beginning of an epoch and is sorted on the smallest name in group. There is a pointer Group which points to an entry in list GROUPS. Lists TASKS and GROUPS are the same in all stations. The stations in the same group always perform the same tasks and broadcast simultaneously. When an epoch ends, then the stations that are in groups on list GROUPS are rearranged into a new list of groups.

Let n be the smallest number such that inequality $n(n+1)/2 \geq |\text{TASKS}|$ holds at the beginning of the epoch. If the number of stations in groups in list GROUPS is at least n, then the epoch is *dense*, otherwise it is *sparse*. Take two stations on positions i and j, respectively, in the sorted list of all the stations in groups in list GROUPS at the end of the previous epoch. They are placed in the same group of the new list if numbers i and j are congruent modulo n. It follows that two groups in list GROUPS differ by at most 1 in their sizes. Initially all stations are considered operational, and positions are meant to be in the sorted list of all names of the stations. This allocation of stations means that when an epoch is sparse, then groups consist of singleton stations, otherwise some groups contain more than one station. The stations are partitioned initially into $\min\{\lceil\sqrt{t}\rceil, p\}$ balanced groups. Ranges are assigned to groups similarly as they are assigned to individual stations in algorithm TWO-LISTS. This determines the initial value of pointer Task_To_Do$_v$ for each station v.

The way the lists are updated after a scheduled broadcast depends on whether a broadcast was performed or not, which is detected by receiving either a message or a signal of collision. When a broadcast takes place, then the tasks performed by the group pointed to by `Transmit`, which was just heard on the channel, are removed from `TASKS`. If a broadcast did not occur, then all the stations in the group pointed to by `Group` are removed from list `GROUPS`. The rules for updating pointers are similar as in algorithm TWO-LISTS. If an item, which is pointed at by the pointer associated with a list, is removed from the list, then the pointer is automatically advanced to the following item.

Lemma 4 *Algorithm* GROUPS-TOGETHER *is reliable.*

Proof The copy of list `GROUPS` is initialized to the same sorted list of all stations in every station. The same entry from these lists is removed after a missing broadcast. When an epoch ends, then copies of this list are identical in all stations, and remain so after they are rearranged. This assures that both list `GROUPS` and pointer `Transmit` are common knowledge, which guarantees that exactly one group of stations is scheduled to broadcast at a phase.

Assume that some station, say, v never crashes. Any scheduled transmission by the group of v is always performed, which results in either a transmission by v or collision heard on the channel. Station v performs at least one new task in each phase, as specified by the first instruction of Round 1 in Fig. 4. It follows that all stations remove at least one entry from their copies of list `TASKS` between two consecutive broadcasts of v.

For any task x, station v eventually either performs x itself or hears about x as performed. This means that eventually list $\texttt{DONE}_v$ contains all the tasks. Now just one broadcast by v results in all stations halting, unless they halted before. □

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

Proof We consider sparse and dense epochs separately. Consider first the sparse epochs of an execution. The amount of work contributed in such noisy phases is $\mathcal{O}(t)$, by an argument similar to that given in the proof of Theorem 1.

Next consider silent phases in sparse epochs. Let p_i denote the number of operational stations and t_i denote the number of outstanding tasks in the beginning of the i-th sparse epoch. We have $p_i = \mathcal{O}(\sqrt{t_i})$, by the definition of a sparse epoch. The amount of work contributed by one phase in the i-th sparse epoch is $\mathcal{O}(p_i) = \mathcal{O}(\sqrt{t_i}) = \mathcal{O}(\sqrt{t})$. Silence at a phase is due to all the stations scheduled to transmit at the phase having crashed. No station scheduled to broadcast in a silent phase is included in list `GROUPS` of the next epoch. Hence the total work in sparse epochs during silent phases is $\mathcal{O}(p\sqrt{t})$.

Next consider dense epochs. Let p_i denote the number of operational stations, t_i denote the number of outstanding tasks, and g_i denote the number of groups in list `GROUPS`, at the beginning of the i-th dense epoch. Observe that $g_i = \mathcal{O}(\sqrt{t_i})$ by the definition of dense epoch. Hence the amount of work during the considered epochs is

$$\mathcal{O}\left(\sum_{i\ge 1} p_i \times g_i\right) = \mathcal{O}\left(\sum_{i\ge 1} p_i \times \sqrt{t_i}\right) . \tag{3}$$

Let s_i be the number of silent phases in the i-th dense epoch. We consider two cases determined by which of the inequalities $s_i < g_i/2$ or $s_i \ge g_i/2$ holds. In the former case, the number of tasks announced as performed on the channel is $\Omega(t_i)$, similarly as in the proof of Theorem 1. It follows that $t_{i+1} \le c_1 t_i$, for some constant value $c_1 < 1$. In the latter case, the number of stations removed from list `GROUPS` in the i-th dense epoch it at least the number s_i multiplied by the minimum size of a group. The size of a group in the i-th dense epoch is $\Theta(p_i/\sqrt{t_i})$. Hence the number of stations removed from list `GROUPS` in the i-th considered epoch is

$$\Omega\left(s_i \times \frac{p_i}{\sqrt{t_i}}\right) = \Omega\left(\sqrt{t_i} \times \frac{p_i}{\sqrt{t_i}}\right) = \Omega(p_i) .$$

This means that $p_{i+1} \le c_2 p_i$, for some constant value $c_2 < 1$.

Every dense epoch falls into one of the two categories discussed above. Therefore either the inequality $t_{i+1} \le c_1 t_i$ holds or $p_{i+1} \le c_2 p_i$ does. In both cases the estimate

$$p_{i+1}\sqrt{t_{i+1}} \le c_3 p_i \sqrt{t_i}$$

is valid, for $c_3 = \min\{c_1, \sqrt{c_2}\} < 1$. Therefore $p_i\sqrt{t_i} \le c_3^{i-1} p_1 \sqrt{t_1}$, for $i > 1$, with $p_1 \le p$ and $t_1 \le t$. Plugging in these estimates into the right-hand side of (3), we obtain

$$p_1\sqrt{t_1}\sum_{i\ge 1} c_3^{i-1} = \mathcal{O}(p\sqrt{t})$$

as the estimate on work accrued in dense phases.

To summarize, we showed that noisy sparse epochs contribute $\mathcal{O}(t)$ to work and silent sparse epochs contribute $\mathcal{O}(p\sqrt{t})$. Dense epochs contribute $\mathcal{O}(p\sqrt{t})$ to work, which completes the proof. □

Corollary 2 *Algorithm* GROUPS-TOGETHER *cannot be beaten in asymptotic work efficiency, by a randomized reliable algorithm for the channel with collision detection, in any adversarial model.*

Proof Combine Theorem 3 with Lemma 2. The former holds for the strongest adversary who is adaptive and may crash up to $p-1$ stations. The latter gives a bound that holds for the weakest adversary who does not fail any station. □

5 Randomized solutions

Corollary 2 states that randomization does not help against *any* adversary to improve efficiency of deterministic solutions for the *Do-All* problem in the model with collision detection. What we showed in Sect. 3, about the model without collision detection, was that randomization does

```
– initialize STATIONS to a sorted list of all $p$ names of stations
– initialize integer variable counter := $p$
– if $t < p^2$ then
    repeat $\lceil\sqrt{t}\,\rceil$ times
      – if $v$ has not been moved to the front of list STATIONS yet
        then toss a coin with the probability counter$^{-1}$ of heads to come up
      – if heads came up in the previous step
        then broadcast $v$ via the channel and attempt to receive a message
      – if some station name $w$ was heard then
          – move station $w$ to the front of list STATIONS
          – decrement counter by 1
– execute Two-Lists with the current order of stations on list STATIONS
```

Fig. 5 Algorithm Mix-Rand; code for station v

not help against strongly-adaptive size-bounded adversaries. The lower bound of Theorem 2 relies on the power of such an adversary. In this section we show that, as far as the channel *without* collision detection is concerned, the power of a size-bounded adversary affects optimal efficiency of deterministic algorithms versus randomized ones.

We present a randomized algorithm Mix-Rand described in Fig. 5. It calls algorithm Two-Lists at the end. In particular, the algorithm uses the same lists and pointers as algorithm Two-Lists. The repeat loop of the algorithm is called *moving to front*. One iteration of this loop is a *phase* of this preprocessing operation. If the name of a station is heard on the channel during moving to front, then the station is said to have been *moved to front*.

Theorem 4 *Algorithm* Mix-Rand *is reliable.*

Proof The algorithm rearranges stations on list STATIONS and next runs algorithm Two-Lists with list STATIONS in this order. The reliability of algorithm Two-Lists does not depend on the initial ordering of list STATIONS, as long as this ordering is the same in all the stations. This property holds because a station is moved to front only after its name was heard on the channel. □

Theorem 5 *Algorithm* Mix-Rand *solves* Do-All *in the channel without collision detection with the minimal expected work $\mathcal{O}(t + p\sqrt{t})$ against the Weakly-Adaptive Linearly-Bounded adversary.*

Proof An execution starts with choosing some cp stations as possible to crash at any time, where $0 < c < 1$ is a constant. Call the selected stations simply *faulty*, since they can crash at any time. There are cp faulty stations and $(1 - c)p$ non-faulty ones.

Consider first the case $p^2 \le t$. The amount of work performed by algorithm Two-Lists is $\mathcal{O}(t + p\sqrt{t} + p\min\{f, t\})$ by Theorem 2. If $p^2 \le t$, then $p = \mathcal{O}(\sqrt{t})$ and hence $f = cp = \mathcal{O}(\sqrt{t})$. Therefore $p\min\{f, t\} = \mathcal{O}(p\sqrt{t})$ and the work of all stations while running Mix-Rand is $\mathcal{O}(t + p\sqrt{t})$.

Consider next the case $p^2 > t$. The non-faulty stations moved to front are called *helpers*. We first show that the number of helpers in an execution of the algorithm is $\Omega(\sqrt{t})$ with a large probability.

During moving to front, a successful transmission is heard at a round with the probability of $(1 - \frac{1}{n})^{n-1} > 1/e$, where n is the number of stations that have not been moved to front and e is the base of the natural logarithm. The first station moved to front is non-faulty with the probability $(1 - c)$. The problem we encounter is that the probability that a new helper is moved to front, at a single phase of moving to front, depends on how many faulty and non-faulty stations have already been moved to front. In the beginning of moving, the probability is at least $(c - 1)/e$. If few non-faulty stations have been moved to front, then the probability of adding a new helper may become larger, while when many helpers have already been moved, then the probability may decrease. We estimate these probabilities uniformly depending on whether at least half of all the non-faulty stations have been moved to front.

Let X be a random variable equal to the number of successes in a sequence of $\lceil\sqrt{t}\rceil$ independent Poisson trials, where the probability s_i of success at the i-th trial is defined as follows. If the number of non-faulty stations moved to front by the i-th phase of moving to front is smaller than $(c-1)p/2$, then $s_i = (c-1)/(2e)$, otherwise $s_i = 1$. For any integer $k \le (c - 1)\lceil\sqrt{t}\rceil/2$, the probability that a total of at least k non-faulty stations are moved to front is not smaller than the probability of the event $X \ge k$.

We use the following Chernoff bound [39] for a sequence of Poisson trials. If Y is the number of successes, μ is the expected value and $0 < \epsilon < 1$, then the inequality

$$\Pr[Y \le (1 - \epsilon)\mu] \le e^{-\epsilon^2\mu/2}$$

holds.

We apply the inequality to variable X. Observe that the expected value μ of random variable X is at least $(c - 1)\lceil\sqrt{t}\rceil/(2e)$, to obtain from the Chernoff bound that the number q of helpers is $\Omega(\sqrt{t})$ with the probability of at least $1 - e^{-a\sqrt{t}}$, for some constant $a > 0$.

Next we consider the amounts of work depending on whether the event that there are $\Omega(t)$ helpers holds, for some specific constant in the Omega notation following from the

Chernoff bound. When the event does not hold, then the work is $\mathcal{O}(pt)$. Suppose that indeed there are $\Omega(t)$ helpers.

When all the helpers are scheduled to broadcast in an epoch, then the number of tasks announced via the channel as performed in the epoch is at least $q(q+1)/2 = \Omega(t)$. There are $\mathcal{O}(1)$ such epochs, hence the total work by the last such an epoch is $\mathcal{O}(p\sqrt{t})$. Consider the first epoch after these when all the helpers are scheduled to transmit. Suppose that there are still tasks in list TASKS. Starting from this epoch, only an initial segment of helpers is used in transmissions.

Observe that the amount of *failing work*, contributed in phases when stations moved to front crashed before the scheduled rounds to transmit, is at most $pq = \mathcal{O}(p\sqrt{t})$. Sparse epochs contribute work that can be charged to $\mathcal{O}(t)$, at phases with successful transmissions, or to failing work at the remaining phases. The amount of work accrued during a dense epoch is $\mathcal{O}(p\sqrt{t})$. A dense epoch has the minimal number of tasks announced via the channel as performed when helpers transmit in the initial segment of the phases in the epoch, followed by phases contributing to failing work. When transmissions by helpers make at least half of a dense epoch, then the number of tasks announced via the channel is $\Omega(t)$. Otherwise the number of faulty stations interspersed between the helpers that are removed from STATIONS is $\Omega(q)$. Since there are q faulty stations between the helpers and there are t tasks to perform, we obtain that there are $\mathcal{O}(1)$ dense epochs, which contribute the amount of work that is $\mathcal{O}(p\sqrt{t})$. It follows that the total amount of work, when there are $\Omega(\sqrt{t})$ helpers, is $\mathcal{O}(t + p\sqrt{t})$.

The total expected work is thus of order

$$(1 - e^{-a\sqrt{t}})\,\mathcal{O}(t + p\sqrt{t}) + e^{-a\sqrt{t}}\,\mathcal{O}(pt)\ .$$

This can be further estimated as follows:

$$\begin{aligned}&\mathcal{O}(t + p\sqrt{t}) + \mathcal{O}(pte^{-a\sqrt{t}})\\ &\quad= \mathcal{O}(t + p\sqrt{t}) + \mathcal{O}(pe^{-a\sqrt{t}+\ln t})\\ &\quad= \mathcal{O}(t + p\sqrt{t}) + \mathcal{O}(p),\end{aligned}$$

which is $\mathcal{O}(t + p\sqrt{t})$. □

We show the following general lower bound about weakly-adaptive adversaries.

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* $\Omega(t + p\sqrt{t} + p\min\{f/(p-f), t\})$.

Proof Part $\Omega(t+p\sqrt{t})$ follows from Lemma 2. We show the remaining one. Let number $g \le f$ be a parameter, we will set its value later in the proof. Consider the following experiment. The algorithm is run for g rounds and any station that wants to perform a transmission such that the broadcast would be successful is failed just before it is to transmit. Additionally, at round g, a sufficient number of the remaining stations, say, those with the smallest names, is failed to make the total number of failures by round g equal to exactly g. We define a probabilistic space in which the elementary events are the sets of names of stations corresponding to sets of size g that can be failed in the experiment. Let $\mathcal{F}$ denote the family of all such elementary events. The probability $\Pr(\omega)$ of $\omega \in \mathcal{F}$ is defined to be equal to the probability of an occurrence of an execution during the experiment in which exactly the stations with names in ω are failed by round g.

The following equality holds

$$\sum_{|A|=p-f}\ \sum_{\omega\cap A=\emptyset} \Pr(\omega) = \binom{p-g}{p-f},$$

where we sum over subsets $A \subseteq [p]$ and elementary events $\omega \in \mathcal{F}$, because each $\Pr(\omega)$ occurs $\binom{p-g}{|A|}$ times on the left-hand side. There are $\binom{p}{p-f}$ subsets $A \subseteq [p]$ with $|A| = p - f$. Hence there is some $C \subseteq [p]$, with $|C| = p - f$, such that the probability that the names of stations failed in the experiment are all outside C is at least

$$\begin{aligned}\binom{p-g}{p-f}\Big/\binom{p}{p-f} &= \frac{(p-g)!}{(f-g)!}\,\frac{f!}{p!}\\ &= \frac{(f+1-g)(f+2-g)\cdots(p-g)}{(f+1)(f+2)\cdots p)}\\ &= \left(1-\frac{g}{f+1}\right)\left(1-\frac{g}{f+2}\right)\cdots\left(1-\frac{g}{p}\right)\\ &\ge \left(1-\frac{g}{f+1}\right)^{p-f}\\ &= \left[\left(1-\frac{g}{f+1}\right)^{(f+1)/g}\right]^{g(p-f)/(f+1)}\\ &\ge 4^{-g(p-f)/(f+1)}\\ &= \Omega(1),\end{aligned}$$

provided $g(p-f)/(f+1) = \mathcal{O}(1)$, which is the case if g equals $g^* = \lceil f/(p-f)\rceil \le f$. Let the adversary declare exactly the stations not in C as prone to failures. Suppose the algorithm is run for $\min\{g^*, t\}/4$ rounds and each station not in C that is to broadcast successfully is failed just before it attempts to do so. Such an execution is consistent with the power of the adversary. The event that no message is heard during $\min\{g^*, t\}/4$ rounds occurs with the probability that is $\Omega(1)$.

The number of operational stations is $\Omega(p)$. None of these stations may halt by round $\min\{g^*, t\}/4$, because otherwise the algorithm would not be reliable, since this round occurs earlier than $\min\{f, t\}/4$. The expected work in such an execution is thus of order

$$\Omega(1)\Omega(p)\min\{g^*/4, t/4\} = \Omega(p\min\{g^*, t\})\ ,$$

which completes the proof. □

Is there an algorithm that needs to perform only the expected minimal work against such weakly-adaptive adversaries who could fail asymptotically more than a constant fraction of the stations? The next fact is an answer to this question in the negative for certain ranges of the number of failures f, as related to p and t.

Corollary 3 *If $f = p(1-o(1/\sqrt{t}))$ and $t = o(p^2)$, then the Weakly-Adaptive f-Bounded adversary can force any randomized algorithm solving* Do-All *for the channel without collision detection to perform the expected amount of work that is $\omega(t + p\sqrt{t})$.*

Proof The lower bound of Theorem 6 states that the amount of work above minimal is $\Omega(p \min\{f/(p-f), t\})$. There are two cases, depending on which quantity under the minimum operator is smaller. If $t \le f/(p-f)$, then the amount of work is $\Omega(pt) = \omega(t + p\sqrt{t})$, for $p = \omega(1)$ and $t = \omega(1)$. Next consider the other case of $f/(p-f) < t$.

Since $p - f = o(p/\sqrt{t})$, the amount of work above minimal is

$$p\frac{f}{p-f} = \omega\left(pf\frac{\sqrt{t}}{p}\right) = \omega(f\sqrt{t}). \tag{4}$$

Plug in $f = p(1 - o(1/\sqrt{t}))$ into the right-hand side of (4) to obtain estimates

$$\begin{aligned}\omega\left(\left(p - o\left(\frac{p}{\sqrt{t}}\right)\right)\sqrt{t}\right) &= \omega(p\sqrt{t} - o(p)) \\ &= \omega(p\sqrt{t}) = \omega(t + p\sqrt{t}),\end{aligned}$$

by the assumed magnitude $p = \omega(\sqrt{t})$ of the number of stations. □

6 Conclusion

In this paper we study the *Do-All* problem in synchronous networks with a broadcast primitive implemented by the multiple-access channel. We consider relations among the following issues: the impact of availability of collision detection, the power of randomization versus deterministic solutions, and specific adversarial models of crashes.

We observe that simple algorithms make it possible for all stations to maintain common knowledge about crashes and local states of stations. This knowledge can be referred to in code of algorithms, which simplifies their design and exposition, and may be used to save on the amount of information sent via the channel.

Most of the previous research on the multiple-access channel has concerned the issues of stability of protocols handling dynamically generated packets. There have been few static algorithmic problems considered, in which the whole input is provided to the stations in advance. This paper attempts to demonstrate that studying problems like *Do-All* in the distributed environment when broadcasting is implemented by the multiple-access channel provides insights into properties of this environment that are complementary to those obtained when studying the stability of handling dynamically generated packets.

References

1. Dwork, C., Halpern, J., Waarts, O.: Performing work efficiently in the presence of faults. SIAM J. Comput. **27**, 1457–1491 (1998)
2. Chlebus, B.S.: Randomized communication in radio networks, a chapter. In: Pardalos, P.M., Rajasekaran, S., Reif, J.H., Rolim, J.D.P. (eds.), Handbook on Randomized Computing, vol 1, pp. 401–456. Kluwer Academic Publisher, Drodrecht (2001)
3. Gallager, R.G.: A perspective on multiaccess channels. IEEE Trans. Inform. Theor. **31**, 124–142 (1985)
4. Attiya, H., Welch, J.: Distributed Computing: Fundamentals, Simulations, and Advanced Topics, 2nd edn. John Wiley, New York (2004)
5. Kanellakis, P.C., Shvartsman, A.A.: Fault-Tolerant Parallel Computation. Kluwer Academic Publisher, Drodrecht (1997)
6. Chlebus, B.S., De Prisco, R., Shvartsman, A.A.: Performing tasks on synchronous restartable message-passing processors. Distribut. Comput. **14**, 49–64 (2001)
7. Chlebus, B.S., Gąsieniec, L., Kowalski, D.R., Shvartsman, A.A.: Balancing work and communication in robust cooperative computation. In: Proceedings of the 16th International Symposium on Distributed Computing (DISC). LNCS 2508, pp. 295–310 (2002)
8. Chlebus, B.S., Kowalski, D.R.: Randomization helps to perform independent tasks reliably. Random Struct. Algorithms **24**, 11–41 (2004)
9. De Prisco, R., Mayer, A., Yung, M.: Time-optimal message-efficient work performance in the presence of faults. In: Proceedings of the 13th ACM Symposium on Principles of Distributed Computing (PODC), pp. 161–172 (1994)
10. Galil, Z., Mayer, A., Yung, M.: Resolving message complexity of Byzantine agreement and beyond. In: Proceedings of the 36th IEEE Symposium on Foundations of Computer Science (FOCS), pp. 724–733 (1995)
11. Kanellakis, P.C., Shvartsman, A.A.: Efficient parallel algorithms can be made robust. Distribut. Comput. **5**, 201–217 (1992)
12. Georgiou, C., Kowalski, D.R., Shvartsman, A.A.: Efficient gossip and robust distributed computation. In: Proceedings of the 17th International Symposium on Distributed Computing (DISC), LNCS 2848, pp. 224–238 (2003)
13. Kowalski, D.R., Shvartsman, A.A.: Performing work with asynchronous processors: message-delay-sensitive bounds. In: Proceedings of the 22nd ACM Symposium on Principles of Distributed Computing (PODC), pp. 265–274 (2003)
14. Georgiou, C., Russell, A., Shvartsman, A.A.: Work-competitive scheduling for cooperative computing with dynamic groups. SIAM J. Comput. **34**, 848–862 (2005)
15. Georgiou, C., Russell, A., Shvartsman, A.A.: The complexity of synchronous iterative Do-All with crashes. Distribut. Comput. **17**, 47–63 (2004)
16. Fernández, A., Georgiou, C., Russell, A., Shvartsman, A.A.: The Do-All problem with Byzantine processor failures. Theor. Comput. Sci. **333**, 433–454 (2005)
17. Clementi, A.E.F., Monti, A., Silvestri, R.: Optimal F-reliable protocols for the do-all problem on single-hop wireless networks. In: Proceedings of thes 13th International Symposium on Algorithms and Computation (ISAAC), LNCS 2518, pp. 320–331 (2002)
18. Abramson, N.: Development of the Alohanet. IEEE Trans. Inform. Theor. **31**, 119–123 (1985)
19. Metcalfe, R.M., Boggs, D.R.: Ethernet: distributed packet switching for local computer networks. Commun. ACM **19**, 395–404 (1976)
20. Goldberg, L.A., Jerrum, M., Kannan S., Paterson, M.: A bound on the capacity of backoff and acknowledgement-based protocols. SIAM J. Comput. **33**, 313–331 (2004)
21. Goldberg, L.A., MacKenzie, P., Paterson, M., Srinivasan, A.: Contention resolution with constant expected delay. J. ACM **47**, 1048–1096 (2000)
22. Håstad, J., Leighton, T., Rogoff, B.: Analysis of backoff protocols for multiple access channels. SIAM J. Comput. **25**, 740–774 (1996)

23. Raghavan, P., Upfal, E.: Stochastic contention resolution with short delays. SIAM J. Comput. **28**, 709–719 (1998)
24. Willard, D.E.: Log-logarithmic selection resolution protocols in a multiple access channel. SIAM J. Comput. **15**, 468–477 (1986)
25. Kushilevitz, E., Mansour, Y.: An $\Omega(D\log(N/D))$ lower bound for broadcast in radio networks. SIAM J. Comput. **27**, 702–712 (1998)
26. Martel, C.U.: Maximum finding on a multiple access broadcast network. Inform. Process. Lett. **52**, 7–13 (1994)
27. Komlós, J., Greenberg, A.G.: An asymptotically nonadaptive algorithm for conflict resolution in multiple-access channels. IEEE Trans. Inform. Theor. **31**, 303–306 (1985)
28. Kowalski, D.R.: On selection problem in radio networks. In: Proceedings of the 24th ACM Symposium on Principles of Distributed Computing (PODC), pp. 158–166 (2005)
29. Greenberg, A.G., Winograd, S.: A lower bound on the time needed in the worst case to resolve conflicts deterministically in multiple access channels. J. ACM **32**, 589–596 (1985)
30. Jurdziński, T., Kutyłowski, M., Zatopiański, J.: Efficient algorithms for leader election in radio networks. In: Proceedings of the 21th ACM Symposium on Principles of Distributed Computing (PODC), pp. 51–57 (2002)
31. Chlebus, B.S., Gołąb, K., Kowalski, D.R.: Broadcasting spanning forests on a multiple access channel. Theor. Comput. Syst. **36**, 711–733 (2003)
32. Gąsieniec, L., Pelc, A., Peleg, D.: The wakeup problem in synchronous broadcast systems. SIAM J. Discrete Math. **14**, 207–222 (2001)
33. Jurdziński, T., Stachowiak, G.: Probabilistic algorithms for the wakeup problem in single-hop radio networks. In: Proceedings of the 13th Annual International Symposium on Algorithms and Computation (ISAAC), LNCS 2518, pp. 535–549 (2002)
34. Chrobak, M., Gąsieniec, L., Kowalski, D.R.: The wake-up problem in multi-hop radio networks. In: Proceedings of the 15th ACM-SIAM Symposium on Discrete Algorithms (SODA), pp. 985–993 (2004)
35. Indyk, P.: Explicit constructions of selectors and related combinatorial structures, with applications. In: Proceedings of the 13th ACM-SIAM Symposium on Discrete Algorithms (SODA), pp. 697–704 (2002)
36. Chlebus, B.S., Kowalski, D.R.: A better wake-up in radio networks. In: Proceedings of the 23rd ACM Symposium on Principles of Distributed Computing (PODC), pp. 266–274 (2004)
37. Fich, F., Ruppert, E.: Lower bounds in distributed computing. In: Proceedings of the 14th International Symposium on Distributed Computing (DISC), LNCS 1914, pp. 1–28 (2000)
38. Halpern, J.Y., Moses, Y.: Knowledge and common knowledge in a distributed environment. J. ACM **37**, 549–587 (1990)
39. McDiarmid, C.: Concentration. In: Habib, M., McDiarmid, C., Ramirez-Alfonsin, J., Reed, B. (eds.), Probabilistic Methods for Algorithmic Discrete Mathematics, pp. 195–248. Springer-Verlag, Berlin (1998)