The Do-All Problem in Broadcast Networks

Bogdan S. Chlebus
Instytut Informatyki
Uniwersytet Warszawski
Banacha 2
Warszawa 02-097, Poland
chlebus@mimuw.edu.pl

Dariusz R. Kowalski
Instytut Informatyki
Uniwersytet Warszawski
Banacha 2
Warszawa 02-097, Poland
darek@mimuw.edu.pl

Andrzej Lingas
Dept. of Computer Science
Lund University
Box 118
S-221 00 Lund, Sweden
Andrzej.Lingas@cs.lth.se

ABSTRACT

The problem of performing t tasks in a distributed system on p failure-prone processors is one of the fundamental problems in distributed computing. If the tasks are similar and independent and the processors communicate by sending messages then the problem is called *Do-All*. In our work the communication is over a multiple-access channel, and the attached stations may fail by crashing. The measure of performance is work, defined as the number of the available processor steps. Algorithms are required to be reliable in that they perform all the tasks as long as at least one station remains operational. We show that each reliable algorithm always needs to perform at least the minimum amount $\Omega(t+p\sqrt{t})$ of work. We develop an optimal deterministic algorithm for the channel with collision detection performing only the minimum work $\Theta(t+p\sqrt{t})$. Another algorithm is given for the channel without collision detection, it performs work $\mathcal{O}(t+p\sqrt{t}+p\cdot\min\{f,t\})$, where $f<p$ is the number of failures. It is proved to be optimal if the number of faults is the only restriction on the adversary. Finally we consider the question if randomization helps for the channel without collision detection against weaker adversaries. We develop a randomized algorithm which needs to perform only the expected minimum work if the adversary may fail a constant fraction of stations, but it has to select the failure-prone stations prior to the start of an algorithm.

Keywords

distributed algorithm, lower bound, multiple-access channel, fail-stop failures, adversary, independent tasks.

1. INTRODUCTION

We consider a distributed system in which p processors need to perform t tasks. If the processors communicate by exchanging messages, are prone to failures, and the tasks are similar and independent then this problem is called *Do-All*.

In this paper we consider a setting in which the processors are stations communicating over a multiple access channel. The system is synchronized by a global clock. The channel operates according to the following rules: if exactly one station performs a broadcast in a step then the message reaches all the stations, and if more of them broadcast simultaneously in a step then mutual collisions happen and no station successfully receives any of these messages. If the stations attached to the channel do not receive a meaningful message at a step 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 with a *ternary feedback*, because of the 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.

The stations are prone to fail-stop failures. Allowable patterns of failures 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 needs to select a subset of stations which might be failed prior to the start of an algorithm, otherwise it is *strongly adaptive*.

Our results. We consider *Do-All* in the context of a broadcast network. More precisely, the communication is over a multiple-access channel, either with or without collision detection. We consider algorithms that are reliable in the sense of being correct in the worst-case scenario when only one station remains available. We show that the minimum amount $\Omega(t+p\sqrt{t})$ of work has always to be performed by a reliable algorithm. This is an absolute lower bound on work performance of any algorithm, which does not depend on the collision detection, randomization or the power of an adversary. We show that in a channel with collision detection this bound can be attained by a deterministic algorithm against any size-bounded adversary. The situation is more complex in a weaker channel without collision detection. We develop a deterministic algorithm for this channel which performs work $\mathcal{O}(t+p\sqrt{t}+p\cdot\min\{f,t\})$ against f-bounded adversary, even if the number of failures is the only restriction on the power of the adversary. This is shown to be optimal by a matching lower bound. Now it could happen that if we wanted to optimize our solutions against weaker adversaries then part $\mathcal{O}(p \cdot \min\{f, t\})$ in the performance bound could be decreased. This indeed is the case: we show that a randomized algorithm can have the expected minimum work $\Theta(t + p\sqrt{t})$ against certain weakly-adaptive size-bounded adversaries. The conclusion is that randomization helps if collision detection is not available and the adversary is sufficiently restricted. The maximum number of faults when this phenomenon happens is a constant fraction of the number of all the stations. Next we show a lower bound which implies that if only $t = o(p^2)$ then a weakly-adaptive f-bounded adversary with $f = p \cdot (1 - o(1/\sqrt{t}))$ can force any algorithm for the channel without collision detection to perform asymptotically more than the minimum work $\Theta(t + p\sqrt{t})$.

Previous work. The problem *Do-All* was introduced by Dwork, Halpern, and Waarts [12], and investigated in a number of papers [7, 9, 10, 11, 14]. All the previous papers considered networks in which each node can send a message to any subset of nodes in one step. The algorithmic paradigms used included balancing work and checkpointing the progress made. This includes using coordinators, which are designated nodes to collect and disseminate information. Dominant models of failures considered have been those of fail-stop failures. The primary measures of efficiency of algorithms used in [12] were the task-oriented work, in which each performance of a task contributes a unit, and communication measured as the number of point-to-point messages. This paper also proposed the *effort* as a measure of performance, which is work and communication combined; one algorithm presented in [12] has effort $\mathcal{O}(t + p\sqrt{p})$.

The early work assuming fail-stop failures model has concentrated on the adversary who could fail all the stations but one. More recent work concerned optimizing solutions against weaker adversaries, while preserving correctness in the worst-case scenario of arbitrary failure patterns, which guarantee only at least one available processing unit.

De Prisco, Mayer, and Yung [11] were the first to use the available processor steps as the measure of work. They present an algorithm which has work $\mathcal{O}(t+(f+1)p)$ and message complexity $\mathcal{O}((f+1)p)$. Galil, Mayer and Yung [14] improved the message complexity to $\mathcal{O}(fp^{\varepsilon}+\min\{f+1, \log p\}p)$, for any positive ε, while maintaining the same work. This was achieved as a by-product of their work on Byzantine agreement with stop-failures, for which they found a message-optimal solution. Chlebus, De Prisco, and Shvartsman [7] 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 [7] achieves the work performance $\mathcal{O}((t + p\log p + f) \cdot \min\{\log p, \log f\})$, and its message complexity is $\mathcal{O}(t + p\log p + fp)$, against suitably defined adversaries who 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 [10] studied the *Do-All* problem when failure patterns are controlled by weakly-adaptive linearly-bounded adversaries. They developed a randomized algorithm with the expected effort $\mathcal{O}(n\log^* n)$, in the case $n = p = t$, which is asymptotically smaller than a known lower bound $\Omega(n\log n/\log\log n)$ on work of any deterministic algorithm. Recently, strongly-adaptive linearly-bounded adversaries have been studied by Chlebus, Gąsieniec, Kowalski and Shvartsman [9] who developed a deterministic algorithm with the effort $\mathcal{O}(n\log^2 n)$. This is the first algorithm known with a performance bound of the form $\mathcal{O}(n \text{ polylog } n)$ when as many as a linear fraction of processing units can be failed by an adversary, all the previously known algorithms had the performance $\Omega(n^2)$ in such a situation. Note however, that neither work nor communication performances should be used as the only criteria to compare algorithmic solutions of *Do-All*; such algorithms are usually designed for concrete environments and optimized for specific adversaries.

Related work. The multiple-access channel, as considered in this paper, is a special broadcast network ([4, 33]). It may be also interpreted as a single-hop radio network, especially in the context of the relevance of collision detection, see e.g. [6]. Most of the previous research on the multiple-access channel has concerned methods of handling packets which the stations keep receiving and which need to be broadcast on the channel as soon as possible. The packets may be generated dynamically in a possibly irregular way which results in a bursty traffic. Techniques like time-division multiplexing are not efficient then, and a better throughput can be achieved if the control is distributed among the stations. This is done by conflict-resolution protocols, which arbitrate among the stations competing for access to the channel; among the most popular protocols is Aloha and the exponential backoff. If packets are generated dynamically then the basic problem is to have stable protocols, which do not make the channel clogged eventually. Recent work in that direction includes the papers of Goldberg, MacKenzie, Paterson and Srinivasan [16], Håstad, Leighton and Rogoff [20], and Raghavan and Upfal [31]; see also the survey of Gallager [15] for an account of the early research and that of Chlebus [6] for recent developments.

Static problems concern a scenario when input data are allocated at the stations prior to the start of an algorithm. The problem of *selection* concerns the situation when some of the stations hold messages, the goal is to broadcast just any single message successfully. Willard [34] developed protocols solving the problem in the expected time $\mathcal{O}(\log\log n)$ in the channel with collision detection. Kushilevitz and Mansour [27] proved a lower bound $\Omega(\log n)$ for the selection problem if collision detection is not available, which yields an exponential gap between two models for this problem. A related problem of finding maximum among the keys stored in a subset of stations was considered by Martel [28].

There is a related *all-broadcast* problem, in which a subset of k among n stations have messages, all of them need to be sent to the channel successively as soon as possible. Komlós and Greenberg [26] showed how to solve it deterministically in time $\mathcal{O}(k + k\log(n/k))$, where both numbers n and k are known. A lower bound $\Omega(k(\log n)/(\log k))$ was proved by Greenberg and Winograd [18].

Gąsieniec, Pelc and Peleg [17] compared various modes of synchrony in multiple-access channel in the context of the *wakeup* problem, in which the system is started and the time when each station joins is controlled by an adversary, while the goal is to perform a successful broadcast as soon as possible. If the stations have access to a global clock then a wakeup can be realized in the expected time $\mathcal{O}(\log n)$ by a randomized algorithm. If the local clocks are not synchronized, there is a randomized solution working in the expected time $\mathcal{O}(n)$. It was also shown in [17] that deterministic algorithms require time $\Omega(n)$, and that there are deterministic schedules working in time $\mathcal{O}(n \log^2 n)$.

Problem *Do-All* specialized to shared-memory models is called *Write-All*: the operation of reading/writing from a memory cell/register is considered to be an individual task. More precisely, in this problem p failure prone processors need to update t shared memory locations. A solution to this problem can be applied to simulate a step of computation of a shared-memory computer, and thus make its computations resilient to processor faults. The problem *Write-All* was introduced by Kanellakis and Shvartsman [21]. Algorithms for the *Write-All* problem have been developed in a series of papers, including those by Anderson and Woll [3], Buss, Kanellakis, Ragde and Shvartsman [5], Chlebus, Dobrev, Kowalski, Malewicz, Shvartsman, and Vřto [8], Groote, Hesselink, Mauw, and Vermeulen [19], Kedem, Palem, Rabin and Raghunathan [23], Kedem, Palem, Raghunathan and Spirakis [24], Kedem, Palem and Spirakis [25], Martel, Park and Subramonian [29], and Martel and Subramonian [30]. A comprehensive account of algorithms for the *Write-All* problem can be found in a book by Kanellakis and Shvartsman [22].

There is another related problem called *Collect*, it was introduced by Saks, Shavit and Woll [32]. It is about a number of processes, each containing a value in a shared memory register: the goal the processes need to achieve is to learn all the values. A process increases its knowledge in a step by reading a register: then it can add the read value to the contents of its own register. The number of read/write operations is a measure of performance. There is an adversary who controls timing of the processes in an asynchronous computation. Ajtai, Aspnes, Dwork and Waarts [1] showed that the problem for n processes can be solved deterministically with work $\mathcal{O}(n^{3/2} \log n)$, by an adaptation of the algorithm of Anderson and Woll [3]. Aspnes and Hurwood [2] developed a randomized algorithm achieving work $\mathcal{O}(n \log^3 n)$ with high probability. A lower bound $\Omega(n \log n)$ for this problem was given in [32].

2. MODEL

Processing units. There are p stations, each with a unique identifier ID in $[1..p]$. The station $\mathcal{P}$ with identifier $\mathrm{ID}(\mathcal{P}) = i$ is denoted as $\mathcal{P}_i$. The system is synchronized by a global clock.

Communication. Stations communicate by broadcasting on a multiple access channel. This model is also called a single-hop radio network [6], and we often say that a station can *hear* the information it receives from the channel. We assume that all the messages sent on the channel are delivered to all the stations, but if many are sent simultaneously then they interfere with each other and are received as garbled. 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.

If a message sent on the channel is heard by a station then the message is said to be *successfully delivered* to it. This happens if exactly one station broadcasts a message: then it is heard by all the stations by the end of the next step. If no messages are sent then the stations can hear only the *background noise*, which is distinct from any meaningful message. If more than one messages are broadcast simultaneously in a step then a *collision* happens, and no station can hear any of these messages. We consider two models depending on what feedback the stations receive if a collision happens.

Channel without collision detection: each station can hear the background noise.

Channel with collision detection: each station can hear the *interference* noise, which is distinct from the background noise.

Failures. Stations fail by crashing. A station which has not failed yet at a step is said to be *operational* in this step. Failure patterns are generated by adversaries. Adversarial models allowing restarts are not considered [7]. An adversary knows the algorithm against which it competes. An adversary is said to be *adaptive with condition $\mathcal{C}$* if it may make its decisions on-line, the only constraint being that the condition $\mathcal{C}$ has to be satisfied in each execution. We consider the following specific adversaries:

STRONGLY-ADAPTIVE f-BOUNDED: 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, where $0 \le f < p$:

(1) It needs to select f *failure-prone* stations prior to the start of an algorithm;

(2) It may fail only the selected failure-prone stations in the course of an algorithm.

We write simply f-BOUNDED for STRONGLY-ADAPTIVE f-BOUNDED. The adversary UNBOUNDED is the same as (STRONGLY-ADAPTIVE) $(p-1)$-BOUNDED. There is no difference in power between strongly and weakly-adaptive size-bounded adversaries when they compete against deterministic algorithms. The adversary LINEARLY-BOUNDED denotes f-BOUNDED, where $f = c \cdot p$, for a constant $0 \le c < p$. None of the considered adversaries may fail all the stations in an execution of an algorithm.

Complexity measures. The principal measure is *work*, which is defined to be the number of available-processor steps: each station contributes a unit for each step when it is operational, even when idling. To have the measure defined precisely, we need to know when an algorithm starts and when it terminates. The stations are provided with input and the time-step to begin, we start counting the available-processor steps from this moment. Each station may come to a halt state at any time, only when it does so then it stops contributing to work. A station which halted in this way is considered non-faulty, hence halts do not restrict the power of adversaries.

Tasks. There are t tasks given as input. They are known to all the stations. The following three properties of tasks are essential:

similar: each takes one step to perform;

independent: can be performed in any order;

idempotent: can be performed many times and concurrently.

The problem *Do-All* is to perform a given set of tasks with these properties, in a message passing network with processing units prone to failures.

Correctness. We require algorithms to be correct against any strategy of the adversary UNBOUNDED. We call such algorithms reliable. Formally, an algorithm solving the *Do-All* problem is *reliable* if the following two conditions are satisfied:

(1) All the tasks are eventually performed, if at least one station remains non-faulty;

(2) All the stations eventually halt, unless failed.

In proofs of lower bounds, we mean an *execution* of an algorithm to be a sequence of configurations of the system in consecutive steps, including the failure pattern, the messages broadcast on the channel, and the sequence of random bits used by each station [13]. If an execution E' is obtained by modifying the actions of an adversary on some other execution E, then it is assumed that each station in E' receives exactly the same sequence of random bits as in E.

If a reliable algorithm is run then no station halts when there are still tasks that have not been completed yet: the remaining stations may be killed and the halted station will not perform any more tasks. This property can be strengthened quantitatively as follows:

LEMMA 1. *A reliable algorithm, possibly randomized, has to perform work $\Omega(t + p\sqrt{t})$ in each execution, even if no failures happen.*

PROOF. It is sufficient to consider the channel with collision detection, in which algorithms have more information. Part $\Omega(t)$ of the bound follows from the fact that each task has to be performed at least once.

A task α is *confirmed* at step i, in an execution of the algorithm, if either a station broadcasts successfully and it has performed α by step i, or more than one station broadcast simultaneously and all of them, with a possible exception of one station, have performed α by step i. At least half of the stations broadcasting in step i and confirming α have performed it by then, so at most $2i$ tasks can be confirmed in step i. Let E_1 be an execution of the algorithm when no failures happen. Let station $\mathcal{P}$ come to a halt at some step j in E_1.

CLAIM: the tasks not confirmed by step j were performed by $\mathcal{P}$ itself in E_1.

Suppose, on the contrary, that this is not the case, let β be such a task. Consider an execution E_2 obtained by running the algorithm and killing any station that performed β in E_1 just before it was to perform β, and all the remaining stations, except for $\mathcal{P}$, killed at step j. The broadcasts on the channel are the same during the first j steps in E_1 and E_2. Hence all the stations perform the same tasks in E_1 and E_2 till step j. The definition of E_2 is consistent with the power of the adversary UNBOUNDED. The algorithm is not reliable because task β is not performed in E_2 and station $\mathcal{P}$ is operational. This justifies the claim.

We estimate the contribution of station $\mathcal{P}$ to work: The total number of tasks confirmed in E_1 is at most $2(1+2+\ldots+j) = \mathcal{O}(j^2)$. Suppose some t' tasks have been confirmed by step j. The remaining $t - t'$ tasks have been performed by $\mathcal{P}$. The work of $\mathcal{P}$ is at least $\Omega(\sqrt{t'} + (t - t')) = \Omega(\sqrt{t})$. $\square$

The amount of work asymptotically equal to $t+p\sqrt{t}$ is called *minimum*. Lemma 1 shows that the minimum work is an absolute lower bound on the amount of work performed by a reliable algorithm in any scenario. The minimum work is a yardstick we will use in the following sections to measure the performance of algorithms.

3. CHANNEL WITHOUT COLLISION DETECTION

We develop a deterministic algorithm TWOLISTS. The algorithm avoids conflicts between broadcasting stations: at each time-step there is at most one station scheduled to broadcast. A broadcast message may consist of just a single bit, since its only purpose is to confirm that the station is still alive. A station does not need to announce which tasks it performed, since all the stations can compute this themselves.

The stations have the same global knowledge implied by what was broadcast on the channel. Each station maintains two circular lists: `TASKS` and `STATIONS`. The items in `TASKS` are the tasks still not announced on the channel as performed, and the items in `STATIONS` are the stations which either made a broadcast each time they were to do so or were not scheduled to broadcast yet at all. Since these lists are defined by what happened on the channel, they are exactly the same in every station.

There is a pointer associated with each of the lists. Let `Station` denote the station pointed on the list `STATIONS` and `Task` be the task pointed on the list `TASKS`. The pointers provide reference points to assign tasks to stations: `Task` is assigned to `Station`, then the next task on `TASKS` is assigned to the next station on `STATIONS`, and so on in the circular order. Notice that a task can be assigned to a number of stations if the list `STATIONS` winds around the list `TASKS` during the assignment process.

(1) Each station performs:

  a) its assigned task

  b) the first task not performed by it yet located after `Task`

(2) `Counter` is decremented by one; if `Counter` ≤ 0 then the stations halt

(3) If $|$`TASKS`$|$ < `Shift` then `Shift` := $\lceil$`Shift`$/2\rceil$

(4) `Station` performs a broadcast

(5) If a broadcast is heard then

  a) the tasks performed by `Station` are removed from `TASKS`

  b) if `TASKS` is empty then the stations halt

  c) the pointer on `STATIONS` is advanced by one position

 else `Station` is removed from `STATIONS`

(6) The pointer on `TASKS` is advanced by $\lceil\sqrt{\texttt{Shift}}\rceil$ positions

Figure 1: Algorithm TwoLists**: a phase.**

After a scheduled broadcast one of the lists is updated as follows. If a successful broadcast happened then the tasks which have been performed by the broadcasting station are removed from `TASKS`. If a broadcast did not happen then the station which failed to broadcast is removed from the list `STATIONS`. If an item pointed at by the pointer is removed from a list then the pointer is assigned automatically to the next item on the list. Other then that the pointers are updated as follows: pointer `Task` is moved by $\lceil\sqrt{\texttt{Shift}}\rceil$ positions, and pointer `Station` is always advanced by one position.

The variable `Shift` is initialized to $\lceil t/2\rceil$. The lists `TASKS` and `STATIONS` are initialized to contain all the tasks and stations, respectively, ordered by their identifiers. The size of `LIST` is denoted as $|$`LIST`$|$. Variable `Counter` is initialized to t. Algorithm TwoLists is in phases repeated in a loop, a phase is given in Figure 1.

Lemma 2. *Algorithm* TwoLists *is reliable.*

The next theorem shows how the performance of TwoLists degrades gracefully with the number of faults.

Theorem 1. *Algorithm* TwoLists *performs work* $\mathcal{O}(t + p\sqrt{t} + p \cdot \min\{f, t\})$ *against the adversary* f-Bounded*, for* $0 \leq f < p$.

Proof. For the purpose of the argument, the time of computation is partitioned conceptually into *rounds*: during a round the value of `Shift` is constant. If no broadcast is heard at a step then both the work performed by all the stations at this step and the work performed by the station that was to broadcast (since its last successful broadcast) is at most $2p$. There are $\mathcal{O}(\min\{f, t\})$ such rounds, because a station performs a new task in each phase, unless failed.

Consider the work performed by broadcasting stations during a round. The work performed while $|\texttt{STATIONS}|^2 \leq$ `Shift` is estimated as follows. Each task performed and reported was performed only once in that way, and this can be charged to $\mathcal{O}(t)$.

Consider next the work performed while $|\texttt{STATIONS}|^2 >$ `Shift`. A segment of consecutive items on `STATIONS` of size $\lceil\sqrt{\texttt{Shift}}\rceil$, starting from `Station`, perform different tasks in a step. Each of these stations is eventually heard, if not failed, so this work can be charged to $\mathcal{O}(p\sqrt{\texttt{Shift}})$. The values of `Shift` are a geometrically decreasing sequence, hence summing over rounds gives the estimate $\mathcal{O}(p\sqrt{t})$. □

Next we show a matching lower bound.

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

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 1. Consider the case when $\Omega(p \cdot \min\{f, t\})$ is the bound. Let $g = \min\{f, t\}$.

Let E_1 be an execution obtained by running the algorithm and killing any station that wants to broadcast successfully during the first $g/4$ steps. Denote as A the set of stations failed in E_1. The definition of E_1 is consistent with the power of the adversary f-Bounded, since $|A| \leq g/4 \leq f$.

Claim: no station halts by step $g/4$ in the execution E_1.

Suppose, on the contrary, that some station $\mathcal{P}$ halts before step $g/4$ in E_1. We show that algorithm is not reliable. To this end we consider another execution E_2, which can be realized by the adversary Unbounded. Let γ be a task

(1) Each station performs:

a) task assigned to its group

b) the first task after `Task` not performed by it yet

(2) `Counter` is decremented by one; if `Counter` ≤ 0 then the stations halt

(3) If $|$`TASKS`$|$ < `Shift` then `Shift` $:= \lceil$`Shift`$/2\rceil$

(4) Each station in `Group` performs a broadcast

(5) If a broadcast was heard, including a collision, then

a) the tasks performed by `Group` are removed from `TASKS`

b) if `TASKS` is empty then the stations halt

c) the pointer on `GROUPS` is advanced by one position

else `Group` is removed from `GROUPS`

(6) The pointer on `TASKS` is advanced by $\lceil\sqrt{\texttt{Shift}}\rceil$ positions

Figure 2: Algorithm GroupsTogether: a phase.

which is performed in E_1 by at most $pg/(4(t-g/4)) \leq pg/(3t)$ stations except $\mathcal{P}$ during the first $g/4$ steps. It exists because $g \leq t$. Let B be this set of stations, we have $|B| \leq pg/(3t) \leq p/3$. We define operationally a set C of stations as follows. Initially C equals $A \cup B$; notice $|A \cup B| \leq 7p/12$. If there is any station that wants to broadcast during the first $g/4$ steps in 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 $g/4 \leq p/4$ steps of E_1, so $|C| \leq 10p/12 < p$.

Let an execution E_2 be obtained by failing all the stations in C at the start and then running the algorithm. The definition of E_2 is consistent with the power of the adversary Unbounded. There is no broadcast heard in E_2 during the first $g/4$ steps. Therefore each station operational in E_2 behaves in exactly the same way in both E_1 and E_2 during the first $g/4$ steps. Task γ is not performed in execution E_2 by step $g/4$, because the stations in B have been failed and the remaining ones behave as in E_1.

Station $\mathcal{P}$ is not failed during E_2 hence it does the same both in E_1 and in E_2. Consider an execution E_3: it is like E_2 till step $g/4$, then all the stations except $\mathcal{P}$ are failed. The definition of E_3 is consistent with the power of the adversary Unbounded. Station $\mathcal{P}$ is operational but halted and task γ is still outstanding in E_3 at step $g/4$. We conclude that the algorithm is not reliable. This contradiction completes the proof of the claim.

Hence there are at least $p - g/4 = \Omega(p)$ stations still operational and non-halted in step $g/4$ in E_1, and they have generated work $\Omega(pg)$ by then. $\square$

Corollary 1. *Algorithm* TwoLists *is asymptotically optimal, among reliable randomized algorithms for the channel without collision detection, against the f-*Bounded *adversary, for $0 \leq f < p$.*

This shows that randomization does not help against the strongest possible size-bounded adversaries for the channel without collision detection. In Section 5 we show that randomization can make a difference for this channel in weaker adversarial models.

4. CHANNEL WITH COLLISION DETECTION

We develop a deterministic algorithm GroupsTogether. The algorithm is a suitable modification of TwoLists. The main difference is that while TwoLists avoids conflicts by its design, the algorithm in this section uses conflicts aggressively as a hedge against faults. Algorithm GroupsTogether is in phases repeated in a loop, a phase is given if Figure 2.

The algorithm maintains two lists. List `TASKS` is the same as in Section 3. The stations are partitioned into groups, which are maintained as a circular list `GROUPS`. There is a pointer which points to `Group`. The stations in the same group always perform the same tasks and broadcast simultaneously. The tasks are assigned to groups as follows: `Task` is assigned to `Group`, then the next task is assigned to the next group, and so on in the circular orders on both lists. The way the lists are updated after a scheduled broadcast depends on the fact if a broadcast happened, which is detected by receiving either a message or a signal of collision. When a broadcast happens then the tasks performed by the group just heard on the channel are removed from `TASKS`. If a broadcast did not happen then `Group` is removed from `GROUPS`. If an item which is pointed at by the associated pointer is removed from a list then the pointer is automatically advanced to the following item. Other then that the pointers are updated as follows: pointer `Task` is moved by $\lceil\sqrt{\texttt{Shift}}\rceil$ positions, and pointer `Group` is always advanced by one position.

The stations are partitioned initially into $\min\{\lceil\sqrt{t}\rceil, p\}$ balanced groups, this partitioning is never changed in the course

(1) If `STATIONS` is empty then the control is returned to the algorithm

(2) Each station in `STATIONS` tosses a coin with the probability $|\texttt{STATIONS}|^{-1}$ of heads to come up

(3) A station that obtained heads broadcasts its ID on the channel

(4) If number i was heard then station $\mathcal{P}_i$ is removed from `STATIONS` and appended to `SELECTED`

Figure 3: Mixing**: a phase.**

of the algorithm. The lists `TASKS` and `GROUPS` are initialized to contain all the tasks and groups, respectively, and are ordered by the identifiers of items. The variable `Shift` is initialized to $\lceil t/2 \rceil$. Variable `Counter` is initialized to t.

Lemma 3. *Algorithm* GroupsTogether *is reliable.*

Theorem 3. *Algorithm* GroupsTogether *performs only the minimum work* $\Theta(t + p\sqrt{t})$ *against the adversary f-*Bounded*, for* $0 \leq f < p$.

Proof. Rounds are defined as in the proof of Theorem 1. If no broadcast is heard at a step then the work performed by all the stations at this step and the stations that were to broadcast (since their last successful broadcast) is at most $2p$. There are at most $\mathcal{O}(\sqrt{t})$ such rounds because of the total number of groups.

Consider the work performed by broadcasting groups during a round. The work performed while $|\texttt{GROUPS}|^2 \leq |\texttt{TASKS}|$ is estimated as follows. Each task performed and then reported was performed by only one group in that way. If groups contain $\mathcal{O}(1)$ stations then this can be charged to $\mathcal{O}(t)$, otherwise to $\mathcal{O}(p\sqrt{t})$.

Consider next the work performed while $|\texttt{GROUPS}|^2 > |\texttt{TASKS}|$. The number stored in the variable $\lceil\sqrt{\texttt{Shift}}\rceil$ has the property that these many groups in list `GROUPS` perform distinct tasks in a step. Each of these groups was heard, if not failed, so this work can be charged to $\mathcal{O}(p\sqrt{\texttt{Shift}})$. The values of `Shift` make a geometrically decreasing sequence, hence summing over rounds gives the estimate $\mathcal{O}(p\sqrt{t})$. $\square$

The fact that the algorithm GroupsTogether needs to perform only the minimum amount of work has the following two consequences:

Corollary 2. *Algorithm* GroupsTogether *is asymptotically optimal, among reliable randomized algorithms for the channel with collision detection, against the strongest size-bounded adversaries.*

Corollary 3. *Algorithm* GroupsTogether *cannot be beaten in asymptotic work performance by any randomized reliable algorithm in any* weaker *adversarial model in which not all stations can be failed.*

5. RANDOMIZED SOLUTIONS

We have shown that randomization does not help against strongly-adaptive size-bounded adversaries. Corollary 3 implies that this is also the case for the channel with collision detection against weaker adversaries. In this section we show that, as far as the channel *without* collision detection is concerned, the power of a size-bounded adversary does matter if we compare the optimal performance of deterministic algorithms versus randomized ones.

We develop a randomized algorithm MixThenWork. It selects a sufficiently long random set of stations, which then run a suitably modified algorithm TwoLists. The algorithm uses the same lists and pointers as TwoLists. Additionally, there is a cyclic list `SELECTED` of stations, also with a pointer. The process of random selection of stations is performed by procedure Mixing. It iterates phases in a loop, as described in Figure 3.

Next we describe procedure AltTwoLists. It operates by having the stations on list `SELECTED` run TwoLists, with `SELECTED` playing the role of `STATIONS`. The remaining stations in `STATIONS`, if any, do not make any attempts to broadcast then. Instead, they listen to the channel to learn about tasks performed by other stations and keep performing consecutive tasks still in `TASKS`. A task is removed from `TASKS` by a station only if it was performed by a station that managed to broadcast on the channel, otherwise it is just marked as done. This allows to maintain the same items on the instances of the list `TASKS` by all the stations. A station halts when it has all the tasks on its copy of `TASKS` marked as done. Procedure AltTwoLists returns control, and a new iteration starts, as soon as all the stations in `SELECTED` fail to broadcast in their assigned time slots. The whole algorithm MixThenWork is described in Figure 4.

Lemma 4. *Algorithm* MixThenWork *is reliable.*

Theorem 4. *The expected work performed by the algorithm* MixThenWork *against the adversary* Weakly-Adaptive Linearly-Bounded *equals the minimum* $\Theta(t + p\sqrt{t})$.

Proof. If $p^2 < t$ then all the stations are in the list `SELECTED` from the start and all stations execute TwoLists algorithm performing work $\mathcal{O}(t)$.

Iterate in a loop:

(1) If $|\texttt{TASKS}| < |\texttt{STATIONS}|^2$ then $\sqrt{|\texttt{TASKS}|}$ phases of Mixing are performed, else all the stations are moved to SELECTED

(2) Procedure AltTwoLists is called

Figure 4: Algorithm MixThenWork**.**

Consider the case $p^2 \geq t$. A new station is added to the list SELECTED in a phase of Mixing with some constant probability. The number of phases of the first call of Mixing is $\mathcal{O}(\sqrt{t})$. It follows from the definition of the adversarial model and by the Chernoff bound that there are $\Omega(\sqrt{t})$ stations in SELECTED that are not failure-prone, after this first call, with the probability at least $1 - e^{-a\sqrt{t}}$, for some $a > 0$. If this event holds then there is just one iteration of the loop in the algorithm, and the bound $\mathcal{O}(p\sqrt{t})$ on the work follows. Only a constant fraction of $\Theta(\sqrt{t})$ stations executing algorithm TwoLists may fail with the probability $1 - e^{-a\sqrt{t}}$. Hence the work done by these $\Theta(\sqrt{t})$ stations is $\mathcal{O}(t)$ by Theorem 1. In the meantime, as many as $\mathcal{O}(p)$ other stations listening to the channel perform work, which is $\mathcal{O}(p/\sqrt{t})$ times larger than work performed by the selected stations. Hence the total work is $\mathcal{O}(t + (p/\sqrt{t})t) = \mathcal{O}(p\sqrt{t})$ with the respective large probability.

Otherwise the work is $\mathcal{O}(pt)$. The total expected work is thus of order

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

which is within the claimed bound. $\square$

Is there an algorithm that needs to perform only the expected minimum work against such weakly-adaptive adversaries who could fail asymptotically more than a constant fraction of the stations? Corollary 4 answers this question in the negative for certain values of p, f and t. We show the following precise lower bound:

Theorem 5. *The* Weakly-Adaptive f-Bounded *adversary can force any reliable randomized algorithm for 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 1. We show the remaining one. Let number $g \leq f$ be a parameter, we will set its value later in the proof. Consider the following experiment: the algorithm is run for g steps, and each station which wants to broadcast successfully is failed just before it is to do so. Additionally, at step g, a sufficient number of the remaining stations, say, those with the smallest identifiers, is failed to make the total number of failures by step g equal to exactly g. We define a probabilistic space in which the elementary events are the sets of IDs of stations corresponding to sets of size g which 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 IDs in ω are failed by step 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 $A \subseteq [1..p]$ 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 $A \subseteq [1..p]$ with $|A| = p-f$. Hence there is some $C \subseteq [1..p]$, with $|C| = p-f$, such that the probability that the IDs of stations failed in the experiment are all outside C is at least

$$\begin{aligned}\binom{p-g}{p-f}\Big/\binom{p}{p-f} &= \frac{(f+1-g)(f+2-g)\cdot\ldots\cdot(p-g)}{(f+1)(f+2)\cdot\ldots\cdot p)} \\ &= (1-\frac{g}{f+1})(1-\frac{g}{f+2})\cdot\ldots\cdot(1-\frac{g}{p}) \\ &\geq (1-\frac{g}{f+1})^{p-f} \\ &\geq 4^{-g(p-f)/(f+1)} \\ &= \Omega(1) ,\end{aligned}$$

provided $g(p-f)/(f+1) = \mathcal{O}(1)$, which is the case if g equals $h = \lceil f/(p-f)\rceil \leq f$.

Let the adversary declare exactly the stations not in C as prone to failures. Suppose the algorithm is run for $\min\{h,t\}/4$ steps and each station not in C which 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 $\min\{h,t\}/4$ steps has the probability $\Omega(1)$. Simultaneously, the number of operational stations is $\Omega(p)$. None of these stations may halt by step $\min\{h,t\}/4$, by an argument similar to that used in the proof of Theorem 2: the algorithm would prove not to be reliable because this step occurs earlier than $\min\{f,t\}/4$. The expected work in such an execution is thus of order $\Omega(1)\cdot p\min\{h/4, t/4\} = \Omega(p\min\{h,t\})$, which completes the proof. $\square$

Corollary 4. *If* $f = p\cdot(1 - o(1/\sqrt{t}))$ *then the adversary* Weakly-Adaptive f-Bounded *can force any algorithm for the channel without collision detection to perform work* $\omega(t + p\sqrt{t})$, *if only* $t = o(p^2)$.

6. CONCLUSIONS

We study solutions to the *Do-All* problem in the context of synchronous broadcast networks. The questions we ask are as follows: What is the impact of the availability of collision detection? Does randomization help? How does the efficiency of solutions depend on various adversarial models? We show that all these parameters have an impact.

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 quite few static algorithmic problems considered, in which the whole input is provided to the attached stations in advance. The broadcast channel is ubiquitous in local area networks, and all its algorithmic aspects deserve a study, those concerning static problems in particular. This paper attempts to demonstrate that this is the case for the problem *Do-All*.

7. REFERENCES

[1] M. Ajtai, J. Aspnes, C. Dwork, and O. Waarts, A theory of competitive analysis of distributed algorithms, in *Proc. 33rd IEEE Symp. of Foundations of Computer Science*, 1994, pp. 401–411.

[2] J. Aspnes, and W. Hurwood, Spreading rumors rapidly despite an adversary, *J. Algorithms*, 26 (1998) 386–411.

[3] R.J. Anderson, and H. Woll, Algorithms for the certified write-all problem, *SIAM J. Computing*, 26 (1997) 1277-1283.

[4] D. Bertsekas, and R.G. Gallager, "Data Networks," Prentice Hall, 1991.

[5] J. Buss, P.C. Kanellakis, P. Ragde, and A.A. Shvartsman, Parallel algorithms with processor failures and delays, *J. Algorithms*, 20 (1996) 45–86.

[6] B.S. Chlebus, Randomized communication in radio networks, a chapter in "Handbook on Randomized Computing," P.M. Pardalos, S. Rajasekaran, J.H. Reif, and J.D.P. Rolim, (Eds.), Kluwer Academic Publishers, to appear.

[7] B.S. Chlebus, R. De Prisco, and A.A. Shvartsman, Performing tasks on synchronous restartable message-passing processors, *Distributed Computing*, 14 (2001) 49–64.

[8] B.S. Chlebus, S. Dobrev, D.R. Kowalski, G. Malewicz, A.A. Shvartsman, and I. Vřto, Towards practical deterministic write-all algorithms, in *Proc., 13th ACM Symp. on Parallel Algorithms and Architectures,* 2001, Crete, Greece, to appear.

[9] B.S. Chlebus, L. Gąsieniec, D.R. Kowalski, and A.A. Shvartsman, Performing tasks in networks with a constant fraction of non-faulty processors, submitted.

[10] B.S. Chlebus, and D.R. Kowalski, Randomization helps to perform tasks on processors prone to failures, in *Proc. 13th Int. Symposium on Distributed Computing*, 1999, Springer LNCS 1693, pp. 284–296.

[11] R. De Prisco, A. Mayer, and M. Yung, Time-optimal message-efficient work performance in the presence of faults, in *Proc. 13th ACM Symp. on Principles of Distributed Computing*, 1994, pp. 161–172.

[12] C. Dwork, J. Halpern, and O. Waarts, Performing work efficiently in the presence of faults, *SIAM J. on Computing*, 27 (1998) 1457–1491.

[13] F. Fich, and E. Ruppert, Lower bounds in distributed computing, *Proc. of the 14th International Symposium on Distributed Computing*, 2000.

[14] Z. Galil, A. Mayer, and M. Yung, Resolving message complexity of byzantine agreement and beyond, in *Proc. 36th IEEE Symp. on Foundations of Computer Science*, 1995, pp. 724–733.

[15] R.G. Gallager, A perspective on multiaccess channels, *IEEE Trans. on Information Theory*, 31 (1985) 124–142.

[16] L.A. Goldberg, P. MacKenzie, M. Paterson, and A. Srinivasan, Contention resolution with constant expected delay, *J. ACM*, 47 (2000) 1048–1096.

[17] L. Gąsieniec, A. Pelc, and D. Peleg, The wakeup problem in synchronous broadcast system, in *Proc. 19th ACM Symp. on Principles of Distributed Computing*, Portland, Oregon, 2000, pp. 113–122.

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

[19] J.F. Groote, W.H. Hesselink, S. Mauw, and R. Vermeulen, An algorithm for the asynchronous write-all problem based on process collision, *Distributed Computing*, 2001, to appear.

[20] J. Håstad, T. Leighton, and B. Rogoff, Analysis of backoff protocols for multiple access channels, *SIAM J. on Computing*, 25 (1996) 740–774.

[21] P.C. Kanellakis and A.A. Shvartsman, Efficient parallel algorithms can be made robust, *Distributed Computing*, 5 (1992) 201–217.

[22] P.C. Kanellakis and A.A. Shvartsman, "Fault-Tolerant Parallel Computation," Kluwer Academic Publishers, 1997.

[23] Z.M. Kedem, K.V. Palem, M.O. Rabin, and A. Raghunathan, Efficient program transformations for resilient parallel computation via randomization, in *Proc. 24th ACM Symp. on Theory of Computing*, 1992, pp. 306–318.

[24] Z.M. Kedem, K.V. Palem, A. Raghunathan, and P. Spirakis, Combining tentative and definite executions for dependable parallel computing, in *Proc. 23rd ACM Symp. on Theory of Computing*, 1991, pp. 381–390.

[25] Z.M. Kedem, K.V. Palem, and P. Spirakis, Efficient robust parallel computations, in *Proc. 22nd ACM Symp. on Theory of Computing*, 1990, pp. 138–148.

[26] J. Komlós, and A.G. Greenberg, An asymptotically nonadaptive algorithm for conflict resolution in multiple-access channels, *IEEE Trans. on Information Theory*, 31 (1985) 303–306.

[27] E. Kushilevitz, and Y. Mansour, An $\Omega(D \log(N/D))$ lower bound for broadcast in radio networks, *SIAM J. on Computing*, 27 (1998) 702–712.

[28] C.U. Martel, Maximum finding on a multiple access broadcast network, *Information Processing Letters*, 52 (1994) 7–13.

[29] C. Martel, A. Park, and R. Subramonian, Work-optimal asynchronous algorithms for shared memory parallel computers, *SIAM J. Computing*, 21 (1992) 1070–1099.

[30] C. Martel, and R. Subramonian, On the complexity of certified write-all algorithms, *J. Algorithms*, 16 (1994) 361–387.

[31] P. Raghavan, and E. Upfal, Stochastic contention resolution with short delays, *SIAM J. on Computing*, 28 (1998) 709–719.

[32] M. Saks, N. Shavit, and H. Woll, Optimal time randomized consensus - making resilient algorithms fast in practice, in *Proc. 2nd SIAM–ACM Symp. Discrete Algorithms*, San Francisco, California, 1991, pp. 351–362.

[33] A.S. Tanenbaum, "Computer Networks," Prentice Hall, 1996.

[34] D.E. Willard, Log-logarithmic selection resolution protocols in a multiple access channel, *SIAM J. on Computing*, 15 (1986) 468–477.