

Collective Asynchronous Reading with Polylogarithmic Worst-Case Overhead

Bogdan S. Chlebus*
Bogdan.Chlebus@cudenver.edu

Dariusz R. Kowalski†
darek@mpi-sb.mpg.de

Alexander A. Shvartsman‡
aas@cse.uconn.edu

ABSTRACT

The *Collect* problem for an asynchronous shared-memory system has the objective for the processors to learn all values of a collection of shared registers, while minimizing the total number of read and write operations. First abstracted by Saks, Shavit, and Woll [37], *Collect* is among the standard problems in distributed computing. The model consists of n asynchronous processes, each with a single-writer multi-reader register of a polynomial capacity. The best previously known deterministic solution performs $\mathcal{O}(n^{3/2} \log n)$ reads and writes, and it is due to Ajtai, Aspnes, Dwork, and Waarts [3]. This paper presents a new deterministic algorithm that performs $\mathcal{O}(n \log^7 n)$ read/write operations, thus *substantially* improving the best previous upper bound. Using an approach based on epidemic rumor-spreading, the novelty of the new algorithm is in using a family of expander graphs and ensuring that each of the successive groups of processes collect and propagate sufficiently many rumors to the next group. The algorithm is adapted to the *Repeatable Collect* problem, which is an on-line version. The competitive latency of the new algorithm is $\mathcal{O}(\log^7 n)$ vs. the much higher competitive latency $\mathcal{O}(\sqrt{n} \log n)$ given in [3]. A result of independent interest in this paper abstracts a gossiping game that is played on a graph and that gives its payoff in terms of expansion.

Categories and Subject Descriptors: F.2.0 [Analysis of Algorithms and Problem Complexity]: General

General Terms: Algorithms, Performance, Theory

*Department of Computer Science and Engineering, University of Colorado at Denver, Campus Box 109, Denver, CO 80217, USA. The work is supported by the NSF Grant 0310503.

†Max-Planck-Institut für Informatik, Stuhlsatzenhausweg 85, Saarbrücken, 66123 Germany, and Instytut Informatyki, Uniwersytet Warszawski, ul. Banacha 2, Warszawa 02-097, Poland. The work is supported in part by the KBN Grant 4T11C04425 and by the NSF-NATO Award 0209588.

‡Department of Computer Science and Engineering, University of Connecticut, Unit 2155, Storrs, CT 06269, USA, and CSAIL, Massachusetts Institute of Technology, Cambridge, MA 02139, USA. The work is supported by the NSF CAREER Award and NSF Grants 9988304, 0121277, and 0311368.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

STOC'04, June 13–15, 2004, Chicago, Illinois, USA.

Copyright 2004 ACM 1-58113-852-0/04/0006 ...\$5.00.

Keywords: Distributed algorithms, collect, graph expansion

1. INTRODUCTION

We consider a system in which asynchronous processes communicate by reading from and writing to shared-memory registers. Coordination of multiple processes, in gathering values from a set of registers, or in writing values to a set of registers, is among the fundamental cooperation problems in distributed computing. Two such problems have been substantially researched in the past decade: the *Collect* problem deals with reading the contents of a set of registers, and the *Write-All* problem deals with writing to a set of registers. For efficiency, the algorithms are structured so that each process does not perform every read or write by itself. Instead, processes cooperate by sharing information about partial progress. In the case of *Collect*, they share the values of registers that have been already collected. In the case of *Write-All*, they share the information about which registers have already been updated.

The *Collect* problem is stated for n processes, each owning a multi-reader single-writer register, initialized to some private value. The goal of each process is to learn all these values. The problem is trivially solved by each process performing n reads, which requires no cooperation and results in $\Theta(n^2)$ reads in total. A potential for cooperation stems from the fact that the initial value of a register can be distributed and stored in other registers, which act as proxies in disseminating this value. A process accumulates values by *collecting* them. To this end, it reads the contents of another process' register and adds any new information to the contents to its own register. Thus it should be possible for processes to perform a total number of reads that is significantly smaller than $\Theta(n^2)$. The *Repeatable Collect* problem is a dynamic on-line counterpart of *Collect*, in which an execution consists of iterations. A value of a register is said to be *fresh* for a collecting process v , if this value appears in the register after process v has been activated to participate in the current iteration. Freshness of collected values is additionally required in a specification of correctness.

The *Collect* problem was introduced by Shavit [38] and used by Saks, Shavit, and Woll [37]. A *Collect* primitive can be used to solve consensus [37], develop timestamps [19, 20, 21, 25, 38], or implement multi-writer registers [30, 43]. Ajtai, Aspnes, Dwork, and Waarts [3] and Aspnes and Hurwood [7] consider the *Repeatable Collect* problem, in which processes perform a sequence of collects, each time fresh values need to be learned.

Summary of previous research. We now briefly state the results on previous work-efficient solutions for the *Collect* problem (a detailed discussion of the background and other related work concludes this section). Efficiency of algorithms in the considered setting is commonly measured in terms of *work* that accounts for all read/write instructions performed by the processes in solving a

problem. The best deterministic solution for *Write-All* [5] attains work $\mathcal{O}(n^{1+\epsilon})$, for any fixed $\epsilon > 0$. An adaptation of this algorithm to the *Collect* problem [3] has work $\mathcal{O}(n^{3/2} \log n)$, giving the best known deterministic upper bound. In both cases, the gap between the best known deterministic upper and lower bounds is asymptotically bigger than a polylogarithmic factor. A lower bound $\Omega(n \log n)$ on work was stated in [3]. Randomized solutions to *Collect*, like in [7], and *Write-All*, like in [33], exceed linear work by a polylog factor. These randomized solutions have been analyzed in terms of work against the *oblivious* adversaries only, and this is too weak to enable the probabilistic method to yield a deterministic solution with a comparable performance.

The competitive latency of deterministic *Repeatable Collect* in [3] is $\mathcal{O}(\sqrt{n} \log n)$, while for the oblivious randomized *Repeatable Collect* the latency is $\mathcal{O}(\log^3 n)$ [7].

Our results. The summary of the contributions is as follows.

- I. We develop a deterministic algorithm that solves the *Collect* problem of size n using $\mathcal{O}(n \log^7 n)$ reads and writes against any adversary that controls interleaving of events at processes. Ours is the first deterministic solution that exceeds the required linear work lower bound $\Omega(n)$ by only a *polylogarithmic* factor. The best previous deterministic solution is given by Ajtai, Aspnes, Dwork, and Waarts [3], and it has a *polynomial* overhead, resulting in work $\mathcal{O}(n^{3/2} \log n)$.
- II. We extend our static algorithm for *Collect* to solve the *Repeatable Collect* problem. The competitive latency of this algorithm is $\mathcal{O}(\log^7 n)$, which is an improvement over a *polynomial* bound $\mathcal{O}(n^{1/2} \log n)$, the best competitive latency known before, given in [3].
- III. Our *Collect* solution is analyzed by interpreting the flow of information as a rumor-spreading process. To measure the efficiency of this process, we introduce a gossiping game on graphs and analyze its properties in terms of the expansion of the underlying graph. This allows us to obtain *worst-case guarantees* on a rate of spreading a rumor. This result is of an independent interest, with a potential of applications to gossip-based algorithms, like those given in [15, 16, 22], and to fault-tolerance of expanders, following an approach first proposed by Upfal [42].

The novelty of our approach is in analyzing the worst-case behavior of a *deterministic* process of spreading rumors. To circumvent technical obstacles inherent in such analysis, we structure the process to mimic proliferation of information along the edges of an expander graphs, as nodes forward knowledge to their neighbors. We develop tools to estimate the rate of flow of information in such a deterministic rumor-spreading process in expanders. The only previous method to obtain a deterministic sub-quadratic solution, as applied in [3], relied on a *Write-All* solution from [5], which is based on a set of permutations of a low contention. Another approach, of [7], relied on analyzing *Collect* in terms of *randomized* rumor spreading against the oblivious adversary.

The model of computation. For the *Collect* problem in the shared-memory model we consider the setting with n asynchronous processes, where there is a multi-reader single-writer register associated with each process. A process executes a sequence of steps, where each step includes a constant-time local computation, and either a read from an arbitrary register or a write to the process' own register. Reads and writes of registers are the only externally-visible operations. An algorithm specifies how the next read operation depends on the set of processes whose registers' values have been already learnt. An asynchronous execution of an algorithm

depends on the timing of events. An execution can be represented by means of an externally-visible *trace*, specified as an infinite sequence $F = \langle p_0, p_1, p_2, \dots \rangle$, where each p_k is a process identifier. The k th event in the trace is the currently pending read or write operation of the process with the identifier p_k . We consider fair executions, determined by traces in which the identifier of each process occurs infinitely often (see [31]).

The efficiency of algorithms solving the *Collect*, *Repeatable Collect* and *Write-All* problems can be measured in terms of time, in terms of the total number of read and write operations, or in terms of competitiveness. The main performance metric we use here is *work*, defined as the total number of reads and writes in an execution. Given an infinite trace F , the work performed in an execution is equal to the sum, over all the processes, of the number of occurrences of the process identifier by the time the process completes its computation according to the algorithm.

The *competitive latency* of a distributed algorithm is defined by Ajtai, Aspnes, Dwork, and Waarts in [3] to be the ratio of the amount of work that the algorithm performs on a particular sequence of collects to the work by the best possible distributed algorithm performing these collects given the same trace. The paper [3] also defines *collective latency* at a point in time to be the worst-case work required to complete a set of collects that are in progress. They show that if L is a bound on the collective latency of a distributed algorithm then $L/n + 1$ is a bound on the competitive latency of the algorithm.

Additional background. The *Collect* problem was introduced in [37, 38]; subsequent literature includes [2, 3, 6, 7, 8, 9]. Saks, Shavit, and Woll [37] consider an asynchronous rumor-spreading-like algorithm to solve *Collect* where each participating process chooses the next register to be read at random. They gave the expected time bound of $\mathcal{O}(\log^2 n / (\log n - \log f))$. Ajtai, Aspnes, Dwork, and Waarts [3] showed how to adapt the *Write-All* algorithm of Anderson and Woll [5] to a solution of *Collect* that performs $\mathcal{O}(n^{3/2} \log n)$ read/write operations; both of these algorithms are not constructive.

It is often useful to interpret asynchronous computation by referring to an adversary, who controls the timing of processes and the fairness of an execution. A precise definition of the power of the adversary is especially significant for randomized solutions for *Collect* and *Write-All*. If the adversary is required to determine, prior to the start of an execution, the order in which the events are enabled at processes, then such a weak adversary is called *fully oblivious*. An intermediate *contents-oblivious* adversary does not have access to the random bits generated locally by the processes, until these bits are written to a register. Aspnes and Hurwood [7] considered a rumor-spreading game in which a step results in the transfer of knowledge between a pair of processes. Each time a process is enabled for such a transfer, it selects some other process to contact at random. They show that a game completes with $\mathcal{O}(n \log^2 n)$ operations with high probability, if the adversary is content-oblivious. They applied this result to analyze a randomized *Collect* solution that takes $\mathcal{O}(n \log^3 n)$ reads and writes with high probability against the same adversary. The randomized algorithm given in [7] is also analyzed in terms of its competitiveness.

Competitive analysis is concerned with the comparison of the cost of a given algorithm to the cost of an optimal algorithm. It was introduced by Sleator and Tarjan [40], and applied in distributed computing by Ajtai, Aspnes, Dwork, and Waarts [3], Awerbuch, Kutten, and Peleg [11], Bartal, Fiat, and Rabani [13], and Georgiou, Russell, and Shvartsman [23].

An algorithm is *adaptive* if its complexity depends only on the number of participating processes. Such algorithms for *Collect* are

given by Afek, Stupp, and Touitou [2] and Attiya, Fouren, and Gafni [8]. Solutions to *Repeatable Collect* can be used to implement atomic snapshot objects — these are shared objects that allow concurrent reads of segments that are individually written by the participating processes. Such objects have been studied in a variety of settings, for instance by Afek *et al.* [1], Anderson [4], Aspnes and Herlihy [6], and Attiya and Rachman [10].

Other related work. *Collect* may be implemented on top of processes that disseminate information, similar to an epidemic spreading of gossip. Such processes have been studied in applied mathematics [12]. Randomized rumor-spreading algorithms have been investigated by Karp, Schindelhauer, Shenker, and Vöcking [28]. Harchol-Balter, Leighton, and Lewin [24] considered the problem of information exchange when the nodes do not initially know each other, and where gossiping is used for node discovery. Demers *et al.* [18] developed epidemic-like algorithms for updating data bases, where the nodes periodically choose other nodes at random and convey the rumors. Kempe, Kleinberg and Demers [29] gave a gossip-style algorithm in which the nodes learn about the nearest resource location. Van Renesse, Minsky, and Hayden [44] showed how to use gossip to gather information about the occurrence of failures. Chlebus and Kowalski [16] studied gossiping in synchronous networks prone to node failures, and developed algorithms that are both time- and message-efficient. They showed how to apply their gossiping solutions to solve a consensus problem. This was extended by Georgiou, Kowalski, and Shvartsman [22] and applied to a problem of performing independent tasks.

The *Collect* problem is related to the *Write-All* problem, introduced by Kanellakis and Shvartsman [26]. *Collect* is normally considered when registers are large, typically polynomial in n , to be able to store the original values of all registers. In contrast, solutions to *Write-All* are expected to use shared atomic registers of a small size, possibly even constant. There is a large body of literature on fault-tolerant PRAM simulations, most of it based on solutions to *Write-All*, see [17, 27, 32, 33, 39]. Buss, Kanellakis, Ragde, and Shvartsman [14] gave a lower bound $\Omega(n \log n)$ on work and developed a deterministic algorithm with work $\mathcal{O}(n^{\log_2 3})$. A deterministic algorithm of Anderson and Woll [5] attains work $\mathcal{O}(n^{1+\epsilon})$, for any constant $\epsilon > 0$. The randomized algorithm of Martel, Park, and Subramonian [33] performs work $\mathcal{O}(n)$ with high probability on $n/\log n \log^* n$ processors. The probability estimates assume the fully oblivious adversary.

Our algorithm for *Collect* uses graphs with good expansion properties. If expanders are selected randomly, then they have the required expansion properties with high probability. A randomized *Collect* solution could operate by first selecting an expander at random, then proceeding as in the generic algorithm defined in Section 3. This results in a randomized algorithm efficient against the adaptive adversary with high probability (see also [7]). Alternatively, such expanders may be found by an exhaustive search, in exponential time. Constructive expanders that are best for our purposes, were given by Ta-Shma, Umans, and Zuckerman [41], who improved the previous construction of Reingold, Vadhan, and Wigderson [36]. The term “constructive” here means that the neighborhood of a node can be found in time that is polynomial in the maximum degree of the graph and polynomial in $\log n$. The maximum degree of a -expanders in [41] misses the lower bound n/a by a polylogarithmic factor.

Document structure. The rest of the document is structured as follows. In Section 2 we discuss expanders and introduce the gossiping game. In Section 3 we present our *Collect* algorithm. The analysis is given in Sections 4 and 5. In Section 6 we treat the *Repeatable Collect* problem. We conclude in Section 7.

2. EXPANDERS AND GOSSIPING GAME

Processes communicate by reading and writing shared registers. The communication patterns of our algorithm are based on suitable expander graphs. Such graphs have good connectivity properties and can be defined in many equivalent ways, see for instance [36, 41, 45]. Following Pippenger [35], we define an a -expander, for a positive integer a , to be a simple graph $G = (V, E)$ with $|V| = n \geq a$ nodes such that if W_1 and W_2 are any sets of nodes, each of a size at least a , then there is a node v_1 in W_1 and another node v_2 in W_2 such that $\{v_1, v_2\}$ is an edge in E .

FACT 1. *For every sufficiently large constant $d > 0$, and any positive integers a and n , with $a < n$, there exists an a -expander with n nodes and of a maximum degree at most $(dn \lg n)/a$.*

Fact 1 is proved by a standard probabilistic argument, e.g., as given first by Pinsker [34]. If we refer to specific a -expanders of a maximum degree $(dn \lg n)/a$, that exist by Fact 1, then we assume that both the numbers a and n are powers of 2. (The notation $\lg x$ stands for the binary logarithm of x throughout the paper.)

For a set of nodes X in a graph G , the notation $N_G(X)$ denotes the set of all neighbors of nodes in X ; when G is a directed graph then we mean in-neighbors. If X is a singleton, say $X = \{v\}$, then $N_G(X)$ is denoted as $N_G(v)$. The set of all nodes w such that there is a path of length at most i in graph G from the node w to some node in X is denoted $N_G^i(X)$, for a positive integer i . Formally, let $N_G^1(X) = N_G(X)$, and define $N_G^{i+1}(X) = N_G(N_G^i(X)) \cup N_G^i(X)$ recursively.

A *gossiping game* is determined by two numbers n and a , where $a \leq n/4$. Suppose that a and n are both powers of 2, to avoid rounding notation. An instance of a play consists of two moves. Our move determines a simple graph $G = (V, E)$ with $|V| = n$. The opponent is asked to specify a sequence $\langle V_0, \dots, V_{\lg n-1} \rangle$ of sets of nodes of G , each of a size $3a$. These sets do not need to be different or disjoint. To define the payoff function, we first construct an auxiliary directed graph H . The nodes of H are these nodes of G that are in $\bigcup_{i=0}^{\lg n-1} V_i$. The directed edges of H are pairs $\langle v, w \rangle$ that correspond to these edges $\{v, w\} \in E$ for which there exist numbers $0 \leq i < j \leq \lg n - 1$ such that $v \in V_i$ and $w \in V_j$. The *payoff set* X consists of these nodes x in $V_{\lg n-1}$ for which there are more than $2a$ nodes $y \in V_0$ such that for each such y there is a path in H from y to x and of length at most $\lg n - 1$. In other words, the payoff set X consists of all the nodes v of $V_{\lg n-1}$ for which the inequality $|N_H^{\lg n-1}(v) \cap V_0| > 2a$ holds. The *payoff* of an instance of a game is defined to be the size $|X|$ of the payoff set X . The opponent strives to minimize the payoff while giving the sequence $\langle V_0, \dots, V_{\lg n-1} \rangle$.

The intuition behind this definition is as follows. A sequence $\langle V_0, \dots, V_{\lg n-1} \rangle$ of sets, each of a size $3a$, models the history of a computation. The set V_i contains the nodes that send a message to all their neighbors at some phase i of computation. The goal of the opponent is to find a sequence $\langle V_0, \dots, V_{\lg n-1} \rangle$ in a given communication graph such that as few as possible nodes in the last set $V_{\lg n-1}$ gather a significant amount $2a$ of the initial information stored in the first set V_0 .

We analyze the gossiping game for an arbitrary a -expander $G = (V, E)$, where $|V| = n$. Let a sequence $\langle V_i \rangle$, for $0 \leq i \leq \lg a + 1$, consist of subsets of V , and $|V_i| = 3a$ for each $0 \leq i \leq \lg a + 1$.

Suppose that graph G is an a -expander. For a set W of nodes of G , if $|W| \geq a$, then $|N_G(W)| > n - a$. We may strengthen this by *relativizing* to a subset $U \subseteq V$ of nodes: if U and W are two sets of nodes of G , each of at least a elements, then the following inequality holds:

$$|N_G(W) \cap U| > |U| - a.$$

LEMMA 1. If $0 \leq i \leq \lg a$ and a set $W \subseteq \bigcup_{j=\lg a+1-i}^{\lg a+1} V_j$ has size at least a , then the set $N_H(W) \cap V_{\lg a-i}$ is of a size larger than $2a$.

PROOF. For every edge $\langle v, w \rangle \in E$ between the nodes $v \in V_{\lg a-i}$ and $w \in W$, a directed edge $\langle v, w \rangle$ is in graph H , because $w \in V_j$ for some $j > \lg a - i$. By relativizing to $V_{\lg a-i}$, we obtain that the set $N_G(W) \cap V_{\lg a-i}$ has size larger than $3a - a = 2a$. Thus also in H the inequality $|N_H(W) \cap V_{\lg a-i}| > 2a$ holds. \square

LEMMA 2. If a set $W \subseteq V_{\lg a+1}$ has size at least a , then there exists a node $w \in W$ such that the set $N_H^{\lg a+1}(w) \cap V_0$ is of a size larger than $2a$.

PROOF. The set $N_H(W) = N_G(W) \cap V_{\lg a}$ has size larger than $2a$, by Lemma 1. Hence, by the pigeonhole principle, there is a set $W_1 \subseteq W$ of a size $a/2$ such that $N_H(W_1) \cap V_{\lg a}$ is of a size larger than a . We will extend this argument to show the existence of sets of nodes $W_i \subseteq V_{\log a+1}$ for $i = 1, \dots, \lg a$, with the following properties: (a) $|W_i| = \frac{a}{2^i}$, and (b) $|N_H^i(W_i) \cap V_{\log a+1-i}| > a$.

This can be done inductively. The set W_1 has just been shown to exist. Suppose we have a set W_i with the required properties (a)-(b), for $i < \lg a$. Apply Lemma 1 to $W = N_H^i(W_i)$ to obtain that $|N_H^{i+1}(W_i) \cap V_{\log a+1-(i+1)}| > 2a$. By the pigeonhole principle, there is a set $W_{i+1} \subseteq W_i$ such that $|W_{i+1}| = |W_i|/2$ and the inequality $|N_H^{i+1}(W_{i+1}) \cap V_{\log a+1-(i+1)}| > a$ holds. The set $W_{\lg a}$ is comprised of a single element $w \in W$, the set $N_H^{\lg a}(w) \cap V_1$ has size larger than a , and hence, by Lemma 1, the set $N_H^{\lg a+1}(w) \cap V_0$ is of a size larger than $2a$. \square

LEMMA 3. There is a set $Z \subseteq V_{\lg a+1}$ such that $|Z| > 2a$ and for every $v \in Z$ the inequality $|N_H^{\lg a+1}(v) \cap V_0| > 2a$ holds.

PROOF. We will show that there are sets Z_i , for $i = 1, \dots, 2a+1$, with the following properties:

- (a) $|Z_i| = i$;
 - (b) for each $v \in Z_i$ the inequality $|N_H^{\lg a+1}(v) \cap V_0| > 2a$ holds.
- We proceed inductively. By Lemma 2, with $W = V_{\lg a+1}$, there is a node $w \in W$ such that $|N_H^{\lg a+1}(w) \cap V_0| > 2a$. Let us set $Z_1 = \{w\}$. Suppose we have a set Z_i , for some $i < 2a+1$, with the required properties (a)-(b). We show how to extend Z_i to Z_{i+1} with the properties (a)-(b). Notice that the set $V_{\lg a+1} \setminus Z_i$ has size at least a . Lemma 2 applied to this set yields a node $w \in V_{\lg a+1} \setminus Z_i$ such that $|N_H^{\lg a+1}(w) \cap V_0| > 2a$. Define Z_{i+1} to be $Z_i \cup \{w\}$, it has the properties (a)-(b). Finally, we may set $Z = Z_{2a+1}$. \square

THEOREM 1. If the underlying graph is an a -expander, for $a \leq n/4$, then payoff in a gossiping game is always larger than $2a$.

PROOF. Let Z be as in Lemma 3. If $a = n/4$, then $\lg a + 1 = \lg n - 1$ and the set Z is included in the payoff set. Consider the case $a < n/4$, then $\lg a < \lg n - 1$. Let $X \subseteq V_{\lg n-1}$ be the payoff set. Suppose, to the contrary, that $|X| \leq 2a$. Then $|V_{\lg n-1} \setminus X| \geq a$. By the expansion property, there is an edge $\langle v, w \rangle$ in H with $v \in Z$ and $w \in V_{\lg n-1} \setminus X$. Since $v \in N_H(w)$, the following inclusions hold:

$$N_H^{\lg a+1}(v) \cap V_0 \subseteq N(N_H^{\lg a+1}(v)) \cap V_0 \subseteq N_H^{\lg n-1}(w) \cap V_0.$$

The size of $N_H^{\lg a+1}(v) \cap V_0$ is larger than $2a$, because $v \in Z$, hence $w \in X$, which is a contradiction with $w \in V_{\lg n-1} \setminus X$. \square

The following fact is of independent interest. It is a generalization of a result by Upfal [42, Lemma 2], that was applied in [42] to an agreement problem. See also [15, 16, 22] for other extensions and applications.

THEOREM 2. Let G be an a -expander. For any set Y of at least $3a$ nodes of G there is a set $Z \subseteq Y$ of a size at least $2a+1$ such that the subgraph of G induced by Z is of a diameter at most $2 \lg a + 2$.

PROOF. Let us set $V_i = Y$, for $i = 0, \dots, \lg a + 1$, in the gossiping game. Let $Z \subseteq V_{\lg a+1} = Y$ be a set that exists by Lemma 3. Consider two nodes z_1 and z_2 in Z . Each z_i , for $i = 1, 2$, is of distance $\lg a + 1$ from all the nodes in a set $X_i \subseteq V_0$ of a size at least $2a + 1$. Since $|V_0| = 3a$, the intersection $X_1 \cap X_2$ is nonempty. Let some node v be in both X_i , for $i = 1, 2$. Graph H is a directed one, with the edges corresponding to edges of G . The distance from z_1 to z_2 in G is at most the distance from z_1 to v in H plus the distance from v to z_2 in H , which is at most $2 \lg a + 2$. \square

Theorem 2 can be interpreted in terms of fault-tolerance properties of networks with good expansion properties: if few nodes fail, then what remains contains a subnetwork of many nodes that can communicate in logarithmic time *only among themselves*. Notice that Theorem 2 relies on expansion of G only, the node degrees in G are not mentioned.

3. THE COLLECT ALGORITHM

We start by fixing a sequence of graphs G_ℓ , for $\ell = 0, \dots, \lg n - 1$. Each G_ℓ has n nodes and is a 2^ℓ -expander of a maximum degree at most $d \frac{n}{2^\ell} \lg n$, for some fixed constant $d > 0$. Existence of such a sequence of graphs follows from Fact 1. The value of constant d is determined in the course of analysis. The graph G_0 is a complete graph of n nodes. Processes interpret the nodes in these graphs as IDs of processes and their corresponding registers.

A process v uses sequences $\pi_{v,0}, \dots, \pi_{v,\lg n-1}$, where $\pi_{v,\ell}$ is an arbitrary permutation of all the neighbors of v in G_ℓ . Let $s_{v,\ell}$ denote the degree of v in G_ℓ . Each sequence $\pi_{v,\ell}$ contains $s_{v,\ell}$ different registers, for $0 \leq \ell \leq \lg n - 1$, and $s_{v,\ell} \leq \frac{dn}{2^\ell} \lg n$. Process v uses also a sequence σ_v , which is a permutation of all the registers except for the register of v . The sequence σ_v is obtained by first concatenating all sequences $\pi_{v,\lg n-1}, \dots, \pi_{v,0}$, and then pruning the multiple occurrences of a register, while leaving only the first one. We refer to the segment of σ_v containing the part of the original permutation $\pi_{v,\ell}$, but without multiple occurrences, as the ℓ -part of σ_v , for every $\ell = 0, 1, \dots, \lg n - 1$.

Algorithm D-Collect has graph G_ℓ and permutations σ_v and $\pi_{v,\ell}$ embedded into the code of each process v . Process v initializes all α_ℓ to 0, and each $s_{v,\ell}$ is set to the degree of v in graph G_ℓ . The control structure of the algorithm is in Figure 1. The main part of the algorithm is structured as the outer repeat-loop that iterates the inner for-loop of $\lg n - 3$ steps. During the ℓ th iteration of the inner loop, for $1 \leq \ell \leq \lg n - 3$, while in the course of the i th iteration of the outer loop, the process v performs the following two read and one write operations. It reads from the register $\pi_{v,\ell}(i \bmod s_{v,\ell})$, next it reads from the first register in the ordering of the sequence σ_v whose contents have not been learnt yet. Then the process v adds the newly read information to its own register.

We will use the following terminology to describe an execution of any instantiation of the algorithm for a specific trace F . A *position* denotes the number k of the current event occurring at process p_k , as determined by the trace $F = \langle p_k \rangle_{k \geq 0}$. An *iteration* done by a process denotes a single iteration of the inner for loop in Figure 1, during which it performs two reads and one write and modifies its local state according to the outcome of the reads. The ℓ th iteration of the inner loop in the i th iteration of the outer loop is the (i, ℓ) -iteration. All the (i, ℓ) -iterations are said to make the ℓ -layer, for each ℓ such that $1 \leq \ell \leq \lg n - 3$. The algorithm D-Collect is structured in such a way that a process reads the registers

```

repeat
  for  $\ell = 1$  to  $\lg n - 3$  do
    set  $\alpha_\ell = \alpha_\ell + 1 \pmod{s_{v,\ell}}$ 
    read register  $\pi_{v,\ell}(\alpha_\ell)$ 
    read first register in  $\sigma_v$  whose value is not known by  $v$ 
    write all the new information read to the register of  $v$ 
until original register values are stored in the register of  $v$ 

```

Figure 1: The main loop of D-Collect for process v .

of its neighbors in graphs in a circular fashion. More precisely, during any $\frac{dn}{2^\ell} \lg n$ consecutive iterations in an ℓ -layer every process v reads registers of all its neighbors in G_ℓ at least once.

4. EPOCHS AND STAGES

An execution of algorithm D-Collect is determined by a specific trace F . We define partitions of traces into segments, to facilitate measuring progress of collecting. This also results in the respective partitions of executions, and we refer to these segments interchangeably. An execution consists of events. Disjoint segments of consecutive events are called epochs. Epochs are grouped into ℓ -segments, for a number parameter ℓ . Details follow.

We partition the trace into consecutive *epochs*. The position when the k th epoch ends is denoted by τ_k . The sequence of positions $\langle \tau_k \rangle_{k \geq 0}$ is defined inductively as follows. The position τ_0 is set to 0. Suppose that we have the k th epoch already defined, by some position τ_k . For each position $\mu > \tau_k$, and every ℓ such that $1 \leq \ell \leq \lg n - 3$, consider a set $T_\ell(\mu)$ consisting of these processes v that v performs at least $7 \frac{dn}{2^\ell} \lg^2 n$ and less than $14 \frac{dn}{2^\ell} \lg^2 n$ iterations after position τ_k and up to the position μ . Let $T_0(\mu)$ be a set consisting of these processes v that v performs at least $7dn \lg^2 n$ iterations by position μ . Define τ_{k+1} to be the minimum position μ after τ_k such that $T_0(\mu) \neq \emptyset$, if such μ exists, otherwise it is the biggest ℓ , among $1 \leq \ell \leq \lg n - 3$, such that $|T_\ell(\mu)| \geq 3 \cdot 2^\ell$. Accordingly, T_{k+1} denotes the set $T_\ell(\tau_{k+1})$, where ℓ is the biggest index such that $1 \leq \ell \leq \lg n - 3$ and $|T_\ell(\tau_{k+1})| \geq 3 \cdot 2^\ell$, provides there is at least one such an index ℓ , otherwise T_{k+1} denotes the set $T_0(\tau_{k+1})$. If the former case applies, then T_{k+1} is of a size (at least) $3 \cdot 2^\ell$, for $1 \leq \ell \leq \lg n - 3$, then the k th epoch is said to be *ℓ -heavy*. In the latter case, the set T_{k+1} is of a constant size at most 5, and the epoch is said to be *0-heavy*.

LEMMA 4. *The epochs are well defined.*

PROOF. Induction on the number of epoch. Suppose that the position τ_k , for some $k \geq 0$, is well defined. We show that τ_{k+1} exists. By fairness of the execution, there is a position $\mu > \tau_k$ such that the set $T_0(\mu)$ is nonempty. Thus τ_{k+1} is a minimum taken from a nonempty set of nonnegative integers. \square

LEMMA 5. *At most $85dn \lg^3 n$ iterations are performed during one epoch, for sufficiently large n .*

PROOF. Consider an epoch k . Let $X(\ell)$, for $0 \leq \ell \leq \lg n - 3$, denote the set of all the processes v such that v performs at least $7 \frac{3dn}{2^\ell} \lg^2 n$ and less than $14 \frac{3dn}{2^\ell} \lg^2 n$ iterations during epoch k . Let $X(\lg n - 2)$ be the set of processes v such that v performs less than $14 \frac{3dn}{2^{\lg n - 2}} \lg^2 n$ iterations during epoch k .

The sets $X(\ell)$ are all pairwise disjoint, because the ranges in their definitions are such. Every process performs less than $14 \cdot 3dn \lg^2 n$ iterations in one epoch, because otherwise this epoch would have concluded earlier, as soon as each process performed

$7 \cdot 3dn \lg^2 n$ iterations during that epoch. We obtain that the equality $|\bigcup_{\ell=0}^{\lg n - 2} X(\ell)| = n$ holds.

Claim. The inequality $|X(\ell)| < 2^{\ell+1}$ holds, for every ℓ such that $1 \leq \ell \leq \lg n - 3$.

Suppose, to the contrary, that $|X(\ell)| \geq 2^{\ell+1}$. Then there is a set $Y \subseteq X(\ell)$ of a size $2^{\ell+1}$ such that every process $v \in Y$ performs at least $7 \frac{3dn}{2^\ell} \lg^2 n - 1 \geq 7 \frac{3dn}{2^{\ell+1}} \lg^2 n$ iterations by position $\tau_k - 1$, by properties of the nodes in $X(\ell)$. It follows from the definition of epoch k that $\tau_k \leq \tau_k - 1$, which is a contradiction, and this proves the claim.

Since all sets $X(\ell)$ are pairwise disjoint and every process is in some set $X(\ell)$, the number

$$\sum_{\ell=0}^{\lg n - 2} |X(\ell)| \cdot 14 \frac{3dn}{2^\ell} \lg^2 n \quad (1)$$

is an upper bound on the number of all the iterations in epoch k . We have clearly that $|X(0)| \leq 5$ and $|X(\lg n - 2)| \leq n$. Hence the number given by (1) can be upper-bounded by

$$5 \cdot 14 \cdot 3dn \lg^2 n + n \cdot 14 \frac{3dn}{2^{\lg n - 3}} \lg^2 n + \sum_{\ell=1}^{\lg n - 3} 2^{\ell+1} \cdot 14 \frac{3dn}{2^\ell} \lg^2 n.$$

The first two terms are $\mathcal{O}(n \log^2 n)$ and the third term is less than $84dn \lg^3 n$, if $n \geq 2^7$. \square

Now we define *ℓ -stages*, for any ℓ between 0 and $\lg n - 3$. An ℓ -stage consists of a segment of consecutive epochs, possibly of only one epoch. Suppose that epoch x is the last one in the k th ℓ -stage. Then $(k+1)$ th ℓ -stage starts at the beginning of epoch $x+1$ and ends at the end of the first epoch $y > x$ for which the number of processes v , such that v performs at least $7 \frac{dn}{2^\ell} \lg^2 n$ iterations from epoch $x+1$ through epoch y , is at least $3 \cdot 2^\ell$, if $\ell > 0$; otherwise it is at least 1. All ℓ -stages are well defined, since every process performs infinitely many iterations in a trace.

The impact of one ℓ -stage, in terms of work performed by some group of processes of size $3 \cdot 2^\ell$, is similar to the impact of one ℓ -heavy epoch. An advantage to consider epochs is that the work done by some processes may be larger during one stage than during one epoch. An advantage of considering stages is that this allows to partition the execution into ℓ -stages, which is not always possible for ℓ -heavy epochs. Hence, to estimate collecting by some $3 \cdot 2^\ell$ processes, we may use stages, but to analyze the algorithm globally, one may use epochs, so as not to overestimate the work (see Section 5). These two approaches can be combined because in the period when k ℓ -heavy epochs occur there are also at least k ℓ -stages (see Lemma 10).

We will use the following notation and terminology regarding the notion of a stage. A process is *productive* in the k th ℓ -stage if it performs at least $7 \frac{dn}{2^\ell} \lg^2 n$ iterations during this stage, for $1 \leq \ell \leq \lg n - 3$. There are at least $3 \cdot 2^\ell$ processes that are productive for the k th ℓ stage; let $S_\ell(k)$ be a set containing $3 \cdot 2^\ell$ of them. Note that if $7 \frac{dn}{2^\ell} \lg^2 n \geq n$, then during ℓ -stage k every productive process collects the values of all registers, using $7 \frac{dn}{2^\ell} \lg^2 n$ reads (iterations) according to permutation σ_v , because G_0 is a complete graph. We show that, if $7 \frac{dn}{2^\ell} \lg^2 n < n$, then every productive process in the ℓ -stage number $1 + \ell \lg n$ knows the values of all registers.

Consider consecutive ℓ -stages. For every process v productive in the k th such stage, the first $\frac{dn}{2^\ell} \lg n$ iterations in a ℓ -layer are called *learning iterations*. Every iteration from an ℓ -layer, which is not a learning iteration, is called a *promoting iteration*. Notice that during one ℓ -stage k every productive process performs $\frac{dn}{2^\ell} \lg n$ learning iteration, and at least $6 \frac{dn}{2^\ell} \lg n$ promoting iterations. This is

because in every ℓ -stage there are at least $7 \frac{dn}{2^\ell} \lg n$ iterations from an ℓ -layer. Learning iterations are used to get to know the information acquired during ℓ -stages. They are efficient because of good expansion properties of graphs G_ℓ hardwired into the algorithm. The promoting iterations are to complement this by utilizing the permutations σ_v .

In the proof of the following Lemma 6 we apply the machinery of gossiping games to graphs G_ℓ . Let v be a process in $S_\ell(k)$ and let $R_{v,\ell}(k)$ consists of those productive processes in $S_\ell(k - \lg n + 1)$ whose registers by the end of the ℓ -stage $k - \lg n + 1$ are all collected by v during the learning iterations in ℓ -stage k . This process v is a (ℓ, k) -learner if $|R_{v,\ell}(k)| \geq \frac{2}{3}|S_\ell(k - \lg n + 1)|$; which means that process v has read at least $2^{\ell+1}$ registers of processes in $S_\ell(k - \lg n + 1)$ during its learning iterations of ℓ -stage k .

LEMMA 6. *For every ℓ -stage k , where $k \geq \lg n$, the number of (ℓ, k) -learners in $S_\ell(k)$ is larger than $\frac{2}{3}|S_\ell(k)|$.*

PROOF. We consider an instance of the gossiping game determined by the graph G_ℓ and the sets $V_i = S_\ell(k - \lg n + 1 + i)$, for $0 \leq i \leq \lg n - 1$. Graph G_ℓ is an a -expander with $a = 2^\ell$. The sizes $|V_i|$ of the sets V_i are all equal to $3 \cdot 2^\ell = 3a$. A payoff set in this case consists of those processes in $S_\ell(k)$ who are (ℓ, k) -learners. By Theorem 1, the payoff of the game is larger than $2a = 2^{\ell+1} = \frac{2}{3}|S_\ell(k)|$. \square

We say that a register r is *unknown to the process v* at an event in an execution, if the initial value of r is not included in the register owned by v prior to this event, otherwise v *knows* the register r .

LEMMA 7. *For every ℓ , where $1 \leq \ell \leq \lg n - 3$, there exists a set $A \subseteq S_\ell((\ell + 1) \lg n)$ of a size $2^{\ell+1}$ such that every process v in A knows all the registers at the end of ℓ -stage $(\ell + 1) \lg n$, in any fair execution of the algorithm **D-Collect**.*

PROOF. Consider ℓ -stage k and set $W \subseteq S_\ell(k)$; when $W = \{v\}$, then we write v rather than $\{v\}$. We will use the following notation:

- $U(k, W)$ denotes the set of registers r such that r is unknown to *some* process v in W at the end of ℓ -stage k . We also use the notation $u(k, W) = |U(k, W)|$.
- $U^*(k, W)$ denotes the set of registers r such that r is unknown to *every* process in W at the end of ℓ -stage k . We also use the notation $u^*(k, W) = |U^*(k, W)|$.
- $\bar{U}(k, W)$ denotes the set of registers r such that r is unknown to *some* process v in W at the end of the *last learning iteration* of ℓ -stage k . We also use the notation $\bar{u}(k, W) = |\bar{U}(k, W)|$.
- $\bar{U}^*(k, W)$ denotes the set of registers r such that r is unknown to *every* process in W at the end of the *last learning iteration* of the ℓ -stage k . We also use the notation $\bar{u}^*(k, W) = |\bar{U}^*(k, W)|$.

We may assume that $n \geq 4$. Consider a trace F . Notice that at the end of a iteration, the knowledge of process v about the other registers is always recorded in its register, as guaranteed by the write operation performed at the end of every iteration. Consider the first $\ell \lg n + 1$ ℓ -stages of trace F .

We show that the following invariant holds after each ℓ -stage $i \lg n + 1$, for $i = 0, \dots, \ell$:

there exists a *witness set* $W \subseteq S_\ell(i \lg n + 1)$ of a size greater than $2^{\ell+1}$ such that for every $B \subseteq W$ of a size at least 2^ℓ , the inequality $u^*(i \lg n + 1, B) < 2^{\ell-i}$ holds.

Proof of the invariant for $i = 0$. One needs to show that after ℓ -stage 1 we have $u^*(1, B) < 2^\ell$, for every set $B \subseteq S_\ell(1)$ such that $|B| \geq 2^\ell$.

During the first ℓ -stage we consider only the read operations performed by processes $v \in B$ in promoting iterations in ℓ -layers according to permutations σ_v . By definition of ℓ -stage, every $v \in B \subseteq S_\ell(1)$ performs at least $6 \frac{dn}{2^\ell} \lg^2 n$ of such operations. By definition of σ_v , every $v \in B$ reads registers of all its neighbors in graph G_ℓ , during ℓ -stage 1. Since G_ℓ is a 2^ℓ -expander and $|B| \geq 2^\ell$, we obtain $|N_{G_\ell}(B)| > n - 2^\ell$, and hence $u^*(1, B) < 2^\ell$.

Proof of the invariant for $0 < i \leq \ell$. Suppose that the invariant holds for all the integers up to $0 \leq i - 1 < \ell - 1$. We prove the invariant for i . It follows from Lemma 6 for $k = i \lg n + 1$ that there is a set $Z \subseteq S_\ell(i \lg n + 1)$ containing $(\ell, i \lg n + 1)$ -learners and such that $|Z| > 2^{\ell+1}$. It follows that for every process $v \in Z$ we have

$$\bar{u}^*(i \lg n + 1, v) \leq u^*((i - 1) \lg n + 1, R_{v,\ell}((i - 1) \lg n + 1)),$$

where the subset $R_{v,\ell}((i - 1) \lg n + 1) \subseteq S_\ell((i - 1) \lg n + 1)$ is of a size greater than $2^{\ell+1}$. We show that set Z can be taken as a witness set W in the invariant for i .

Let $\bar{W} \subseteq S_\ell((i - 1) \lg n + 1)$ be a witness in the invariant for $i - 1$. We have that

$$R_{v,\ell}((i - 1) \lg n + 1) \cap \bar{W} \subseteq S_\ell((i - 1) \lg n + 1)$$

and that $|R_{v,\ell}((i - 1) \lg n + 1) \cap \bar{W}| > 2^\ell$, for every process $v \in Z$. By the invariant for $i - 1$, we have that the following bounds

$$\begin{aligned} \bar{u}^*(i \lg n + 1, v) &\leq u^*((i - 1) \lg n + 1, R_{v,\ell}((i - 1) \lg n + 1) \cap \bar{W}) \\ &\leq 2^{\ell-i+1} \end{aligned}$$

hold for every $v \in Z$. Consider any $B \subseteq Z$ of a size 2^ℓ and a process $v \in B$ and its pattern of reads during its promoting iterations while in ℓ -stage of number $i \lg n + 1$. We restrict our attention only to read operations performed by processes $v \in Z$ in the iterations in ℓ -layers according to permutations σ_v . By the definition of ℓ -stage, every $v \in B$ performs at least $6 \frac{dn}{2^\ell} \lg^2 n$ such operations: at least $7 \frac{dn}{2^\ell} \lg^2 n - \frac{dn}{2^\ell} \lg^2 n$ iterations are promoting, and in one iteration we perform one read according to σ_v . Every $v \in B$ can read only registers from

$$U = U^*((i - 1) \lg n + 1, R_{v,\ell}((i - 1) \lg n + 1) \cap \bar{W}),$$

whose size is at most $2^{\ell-i+1}$. We need to prove that $|U^*(i \lg n + 1, B)| < 2^{\ell-i}$.

Suppose to the contrary that $U^* = U^*(i \lg n + 1, B)$ is of a size at least $2^{\ell-i}$. Consider set $U \setminus U^*$. Notice that $U^* \subseteq U$, hence $|U \setminus U^*| \leq 2^{\ell-i}$. It follows from the definition of set B that the total number of reads of registers from $U \setminus U^*$ performed by processes from B during their promoting iterations is at least

$$|B| \cdot 6 \frac{dn}{2^\ell} \lg^2 n = 6dn \lg^2 n. \quad (2)$$

Claim 1. Suppose that a set $X_j \subseteq B$, for every $j = 1, \dots, \ell - i$, contains every process $v \in B$ that has read all its neighbors in graph G_j as governed by the permutation σ_v by the end of the promoting iterations of ℓ -stage numbered $i \lg n + 1$. Then $|X_j| < 2^j$, for any such j .

Suppose, to the contrary, that, for some $1 \leq j \leq \ell - i$, the set X_j is of a size at least 2^j . Then $u^*(i \lg n + 1, X_j) < 2^j \leq 2^{\ell-i}$, and hence $u^*(i \lg n + 1, B) \leq u^*(i \lg n + 1, X_j) < 2^{\ell-i}$, which is a contradiction with $|U^*| \geq 2^{\ell-i}$. This proves of Claim 1.

Claim 2. Let $Y_j \subseteq (U \setminus U^*)$ contain every register $v \in (U \setminus U^*)$ that has been read by at least $3 \frac{dn}{2^j} \log n$ but less than $6 \frac{dn}{2^j} \log n$

processes from B , while reading according to the permutation σ_v during the promoting iterations of ℓ -stage number $i \lg n + 1$, for every j between 1 and $(\ell - i - 1)$. Then $|Y_j| \geq 2^j$ for some among these values of j .

Suppose to the contrary that for every j such that $1 \leq j \leq \ell - i - 1$, we have $|Y_j| < 2^j$. Recall that $|U \setminus U^*| \leq 2^{\ell-i}$. It follows that the total number of reads from $U \setminus U^*$ by the processes from B during promoting iterations is less than

$$\sum_{j=1}^{\ell-i-1} |Y_j| \cdot 6 \frac{dn}{2^j} \log n + (|U \setminus U^*| - |Y_{\ell-i-1}|) \cdot 3 \frac{dn}{2^{\ell-i-1}} \log n,$$

which is less than

$$\sum_{j=1}^{\ell-i-1} 2^j \cdot 6 \frac{dn}{2^j} \log n + 2^{\ell-i} \cdot 3 \frac{dn}{2^{\ell-i-1}} \log n = 6(\ell - i)dn \log n < 6dn \log^2 n.$$

This contradicts the lower bound (2), thus proving Claim 2.

Consider a set Y_j , with a property as in Claim 2.

Claim 3. Each register $r \in Y_j$ is read by at least $\frac{dn}{2^j} \log n$ processes $v \in B$, such that v has read the whole j -part of σ_v by the end of ℓ -stage numbered $i \lg n + 1$.

There are at least $3 \frac{dn}{2^j} \log n$ processes from B reading r . The number of those which may read register r before they read the whole j -part of σ_v is at most the sum of degrees of expanders $G_{\lg n - 3}, \dots, G_j$ - since every read of register r is according to some i -part, for $\lg n - 3 \geq i \geq j$, of some σ_v , and this means reading r by the neighbors in these expanders. This is at most

$$\frac{dn}{n/8} \log n + \dots + \frac{dn}{2^j} \log n \leq 2 \frac{dn}{2^j} \log n \text{ processes.}$$

Hence there are at least $3 \frac{dn}{2^j} \log n - 2 \frac{dn}{2^j} \log n = \frac{dn}{2^j} \log n$ processes $v \in B$ which read r and the whole j -part of σ_v by the end of ℓ -stage numbered $i \lg n + 1$. This completes the proof of Claim 3.

By Claims 3 and 2 and the expansion of G_j , we have $|N_{G_j}(Y_j)| > n - 2^j$, which together with the bound $|B| = 2^\ell \geq 2^{j+1}$ imply $|N_{G_j}(Y_j) \cap B| > 2^j$. Since $N_{G_j}(Y_j) \cap B \subseteq X_j$ we obtain that $|X_j| > 2^j$, which contradicts Claim 1. Hence the inequality $|U^*(i \lg n + 1, B)| < 2^{\ell-i}$ holds, which implies the invariant for i .

Deriving the lemma from the invariant. The invariant for $i = \ell \lg n + 1$ immediately yields what we seek to prove in the lemma. Indeed, from the invariant for $i = \ell$ we obtain that there is a set $W \subseteq S_\ell(\ell \lg n + 1)$, of a size $2^{\ell+1}$, such that for every set $B \subseteq W$ of a size 2^ℓ , the inequality $u^*(\ell \lg n + 1, B) < 1$ holds. Let R_v denote $R_{v,\ell}(\ell \lg n + 1)$, for every process v . By Lemma 6, we obtain that there is a set $A \subseteq S_\ell((\ell + 1) \lg n)$ of a size $2^{\ell+1}$ such that every process v from A knows the rumors stored in $U^*(\ell \lg n + 1, R_v)$, where $R_v \subseteq S_\ell(\ell \lg n + 1)$ is of a size $2^{\ell+1}$. Both the sets R_v and W are included in $S_\ell(\ell \lg n + 1)$, while $|S_\ell(\ell \lg n + 1)| = 3 \cdot 2^\ell$, and $|W|, |R_v| \geq 2 \cdot 2^\ell$. Since also $|R_v \cap W| \geq 2^\ell$, we obtain that each process in A may possibly not know the registers only from $U^*(\ell \lg n + 1, W \cap R_v)$, while $u^*(\ell \lg n + 1, W \cap R_v) = u^*(\ell \lg n + 1, W) = 0$. \square

5. COMPLETING THE ANALYSIS

We use the following classification of epochs. An epoch is said to be *a-successful*, for $1 \leq a \leq n$, if there is a set W of processes such that $|W| \geq a$ and every process in W knows the original values of all the registers after the epoch is finished.

The number of fast learners is large. In this section we consider an execution to the first $n/8$ -successful epoch. For a number ℓ , where $1 \leq \ell \leq \lg n - 3$, the ℓ -set consists of these epochs that are 2^ℓ -successful but not $2^{\ell+1}$ -successful.

If i -heavy epoch k is in ℓ -set and $i \geq \ell$ then we call such an epoch *short*, otherwise we call it *long*. An intuition behind this terminology is as follows. If $i \geq \ell$, then the set T_k is large, and every process in T_k performs “few” iterations during such a short epoch k . Otherwise the set T_k is small, and each process in T_k performs “many” iterations during such a long epoch k . The notions of “few” and “many” are defined relative to the quantity $\frac{dn}{2^\ell} \lg^2 n$.

Consider a fair execution of the algorithm D-Collect and the corresponding trace. Let the first 2^ℓ -successful epoch be denoted by k_ℓ , for $1 \leq \ell \leq \lg n - 3$. Suppose that k_ℓ is not $2^{\ell+1}$ -successful. Let \hat{k}_ℓ be the first epoch after k_ℓ such that at least one of the following conditions holds:

Condition 1. There are at least $\log^3 n$ of short epochs among $k_\ell + 1, \dots, \hat{k}_\ell$.

Condition 2. There are at least 2^i processes, for some integer $i > \ell$, that have not completed collecting by the end of epoch k_ℓ , and each such process occurs in at least $\frac{dn}{2^i} \log n$ iterations in the long epochs from $k_\ell + 1$ through \hat{k}_ℓ .

Epoch \hat{k}_ℓ is well defined, provided epoch k_ℓ exists. To see this, suppose, to the contrary, that \hat{k}_ℓ does not exist. Then there are less than $\log^3 n$ short epochs $k > k_\ell$. But every process occurs infinitely many times in the trace, so it occurs also infinitely many times during long epochs $k > k_\ell$, for $\ell \leq \lg n - 1$, which contradicts not holding of *Condition 2*.

In the next two lemmas we estimate the cost of the short and the long epochs from epoch $k_\ell + 1$ through epoch \hat{k}_ℓ .

LEMMA 8. *The number of iterations executed by processes during short epochs, that occur starting from epoch $k_\ell + 1$ through epoch \hat{k}_ℓ , is at most $85dn \log^6 n$, for sufficiently large n .*

PROOF. There are at most $\log^3 n$ short epochs, each of them takes at most $85dn \log^3 n$ iterations, by Lemma 5. \square

LEMMA 9. *The number of iterations executed by processes during long epochs, that occur starting from epoch $k_\ell + 1$ through epoch \hat{k}_ℓ , is at most $88dn \log^3 n$, for sufficiently large n .*

PROOF. Consider all iterations performed by processes during long epochs among $k_\ell + 1, \dots, \hat{k}_\ell - 1$. Let A_i , for every integer $1 \leq i \leq \log n$, denote set of processes such that each executes more than $\frac{dn}{2^i} \log^2 n$ and at most $2 \frac{dn}{2^i} \log^2 n$ iterations during considered long epochs. Let B denote a set of these processes that each of them executes at most $d \log^2 n$ iterations. The following cases are the only logically possible:

Case 1. $i > \ell$.

Since *Condition 2* is not satisfied for epoch $\hat{k}_\ell - 1$, we have that $|A_i| < 2^i$, and the total number of iterations performed by the processes in A_i is at most $2dn \log^2 n$.

Case 2. $i \leq \ell$ and $|A_i| \leq 2^i$.

The total number of iterations performed by processes from A_i is at most $2dn \log^2 n$.

Case 3. $i \leq \ell$ and $|A_i| > 2^i$.

We show that this case actually cannot occur. Suppose to the contrary that $i \leq \ell$ is the integer such that $|A_i| > 2^i$. Notice that during the epochs under consideration each process in A_i executes more than $\frac{dn}{2^i} \log^2 n$ iterations. This is more than $\frac{dn}{2^i} \log n$ iterations in i -layer, while reading registers of its neighbors in G_i . The

neighbors in graphs G_ℓ are read in a circular fashion, hence fewer than $2^i < |A_i|$ processes from A_i may not read a register containing all the initial values during its $\frac{dn}{2^i} \log n$ consecutive iterations from i -layer. Recall that, by the definition of a 2^ℓ -successful epoch, the set of registers containing all the values is of a size at least $2^\ell \geq 2^i$. The remaining processes from A_i have read such a register and halted. This is a contradiction with the fact that all the processes in A_i perform more than $\frac{dn}{2^i} \log^2 n$ iterations, and so more than $\frac{dn}{2^i} \log n$ iterations from an i -layer, during the epochs considered. This completes a proof that Case 3 cannot occur.

It follows that the number of iterations during long epochs, starting at $k_\ell + 1$ through $\hat{k}_\ell - 1$, is less than

$$|B| \cdot d \log^2 n + \sum_{i=1}^{\log n} 2dn \log^2 n \leq dn \log^2 n + 2dn \log^3 n.$$

By Lemma 5, the last epoch \hat{k}_ℓ contains at most $85dn \log^3 n$ iterations, and so the cost of all the long epochs, from $k_\ell + 1$ through \hat{k}_ℓ , is at most $dn \log^2 n + 2dn \log^3 n + 85dn \log^3 n \leq 88dn \log^3 n$, which completes the proof. \square

By the end of epoch \hat{k}_ℓ , at least $2^{\ell+1}$ processes collected all the values.

LEMMA 10. *Epoch \hat{k}_ℓ is $2^{\ell+1}$ -successful.*

PROOF. If Condition 1 is satisfied for epoch \hat{k}_ℓ , then there is an integer $i \geq \ell$ such that there are at least $\log^2 n$ i -heavy epochs among short epochs in the period $k_\ell + 1, \dots, \hat{k}_\ell$, hence also at least $\log^2 n$ i -stages from the beginning of epoch $k_\ell + 1$ to the end of epoch \hat{k}_ℓ . Now it is sufficient to apply Lemma 7 to these i -stages and the assumption $\ell \leq \lg n - 3$.

Otherwise, Condition 2 is satisfied for epoch \hat{k}_ℓ , for a respective number $i > \ell$. Let W denote a set of processes such that $|W| \geq 2^i \geq 2 \cdot 2^\ell$, each process in W has not completed collecting by the end of epoch k_ℓ and it occurs in at least $\frac{dn}{2^i} \log n$ iterations in long epochs among those in the period $k_\ell + 1, \dots, \hat{k}_\ell$. We can consider such i since $\ell \leq \lg n - 1$. Recall that the neighbors of a node in G_i are read in a circular fashion. Fewer than $2^\ell \leq |W|/2$ processes from W may not read a register containing all the values during its $\frac{dn}{2^i} \log n$ consecutive iterations, by properties of the expander G_i . By the definition of a 2^ℓ -successful epoch, the set of registers containing all the values is of a size at least $2^\ell > 2^i$. There are at least $|W|/2 \geq 2^\ell$ of the remaining processes. Each of them has read such a register and has recorded in its own register the complete set of values. \square

Slow learners get informed inexpensively. We now assess the work of fast and slow learners.

LEMMA 11. *The work accrued after the first iteration when at least $n/8$ processes collected all the initial values of registers each, is $\mathcal{O}(n \log^3 n)$.*

PROOF. Let τ be the first $n/8$ successful epoch. Let P denote set of unsuccessful processes at the end of epoch τ . For every non-successful process $v \in P$ consider its first $2dn \log^2 n$ iterations after the end of epoch τ , partitioned into $\lg n - 2$ intervals, where i th interval has $\frac{dn}{2^i} \log^2 n$ iterations, for $3 \leq i \leq \lg n$, and $(\lg n + 1)$ st interval contains the remaining iterations. Note that we number intervals from 3 to $\lg n + 1$. Let P_i , for $3 \leq i \leq \lg n$, denote the set of processes which are still unsuccessful after performing its i th interval of iterations.

Claim. $|P_i| < n/2^i$, for every $3 \leq i \leq \lg n$.

Suppose, to the contrary, that $|P_i| \geq n/2^i$, for some $3 \leq i \leq \lg n$. In the i th interval each node $v \in P_i$ performed at least $d2^i \log n$ iterations from $(\lg n - i)$ -layer, which means that it read all registers of its neighbors in $n/2^i$ -expander graph $G_{\lg n - i}$. By expansion property of this graph and assumption $|P_i| \geq n/2^i$ we have that less than $n/2^i$ processes in P_i did not read from register of previously successful process (there are at least $n/8 \geq n/2^i$ successful processes after epoch τ). This is a contradiction, since there is a process $v \in P_i$ which is successful by the end of its i th interval. This completes the proof of the claim.

It follows from the Claim that each process is successful by the end of its $\lg n$ interval. It also follows that the work done by the moment where all processes are successful is at most $\sum_{i=3}^{\lg n} |P_i| \cdot d2^i \log^2 n < dn \log^3 n$. \square

THEOREM 3. *Algorithm D-Collect solves any instance of the Collect problem of size n with a total of $\mathcal{O}(n \log^7 n)$ read and write operations.*

PROOF. Consider a number ℓ such that $0 \leq \ell \leq \lg n - 3$. If epoch k_ℓ is 2^ℓ -successful, for some $i > \ell$, then we do not need to consider progress achieved during the epochs in ℓ -set, which is actually an empty set. Let us assume that epoch k_ℓ is not $2^{\ell+1}$ -successful. The inequality $k_{\ell+1} \leq \hat{k}_\ell$ holds, by Lemma 10. The work accrued during the epochs starting at $k_\ell + 1$ through $k_{\ell+1}$ is at most

$$85dn \log^6 n + 88dn \log^3 n \leq 90dn \log^6 n,$$

by Lemmas 8 and 9, for sufficiently large n . There are at most $\lg n - 2$ possible values of ℓ , hence the work spent by the moment when at least $n/8$ processes gathered all registers each is at most

$$\log n \cdot 90dn \log^6 n + \mathcal{O}(n \log^3 n) = \mathcal{O}(n \log^7 n).$$

By Lemma 11, the remaining work is $\mathcal{O}(n \log^3 n)$, hence the total work is $\mathcal{O}(n \log^7 n)$. \square

6. REPEATABLE COLLECT

Repeatable Collect is an on-line dynamic version of the *Collect* problem, and it was defined in [3, 7]. We modify the (static) collect algorithm to obtain the *Repeatable Collect* solution that we call algorithm D-ReCollect.

Algorithm D-ReCollect. We obtain algorithm D-ReCollect by modifying algorithm D-Collect as follows.

Each process v has a *timestamp* that consists of an array α_v of integers and an array $rumors_v$ of lists of rumors. Initially, $\alpha_v(v)$ is a number of the current iteration, and $\alpha_v(w) = 0$ for every $w \neq v$. Also initially $rumors_v(v)$ contains only the current value of the register of v , and $rumors_v(w)$ are empty lists, for all $w \neq v$.

The algorithm collects values by using different updating and selecting rules.

1. It uses the list $rumors_v(v)$, as the list of known rumors, to select a register according to permutation σ_v .
2. If v reads a register of w according to σ_v , then it adds only the current value of this register, rather than all the values of the other registers, as a rumor of w to all its lists $rumors_v(z)$, for all z .
3. If it reads a timestamp, from a register of some process w , according to permutation $\pi_{v,w}$, then it updates its α_v and $rumors_v$ as follows, for every entry z . If $\alpha_w(z) = \alpha_v(z)$

then v sets $rumors_v(z) := rumors_v(z) \cup rumors_w(z)$. If $\alpha_w(z) > \alpha_v(z)$ then v sets $rumors_v(z) := rumors_w(z)$. Finally, $\alpha_v(z)$ is set to $\max\{\alpha_v(z), \alpha_w(z)\}$.

If we add some rumor of process w to the list $rumors_v(z)$, but there is already some rumor of w there, then we replace the old one by the new one, so that each list $rumors_v(z)$ contains at most n rumors, each of a different process. Process v stops the current iteration exactly when $rumors_v(v)$ attains size n .

LEMMA 12. *Algorithm D-ReCollect is correct.*

PROOF. We need to show that each rumor stored in $rumors_v(v)$ is fresh. To show this we prove a stronger invariant:

in any state of execution, and for any active processes v and w , if $\alpha_v(v) = \alpha_w(v)$, then all the rumors in $rumors_w(v)$ are fresh according to v .

We argue in terms of iterations, each consisting of three operations. We treat trace F as a sequence of iterations that is ordered by ends of iterations. Before the first iteration this is true, by initialization rules. Assume that invariant holds by iteration τ . We prove that it holds for iteration $\tau + 1$. Suppose that this is an iteration of process v . In this iteration v reads a register of some process w according to $\pi_{v,t}$ and register of some w' according to σ_v . Process v may change its list $rumors_v(z)$, for any process z , depending which of the following cases is applicable.

Case 1: $\alpha_w(z) = \alpha_v(z)$.

Process v adds some new rumors from $rumors_w(z)$. If additionally $\alpha_z(z) = \alpha_v(z)$ then, by the invariant, those added rumors are fresh, so the new $rumors_v(z)$ contains only fresh rumors according to z . Otherwise we are safe since the assumption $\alpha_v(z) = \alpha_z(z)$ in the invariant is not satisfied.

Case 2: $\alpha_w(z) > \alpha_v(z)$.

Process v changes $\alpha_v(z) := \alpha_w(z)$ and sets its $rumors_v(z)$ to be $rumors_w(z)$. If additionally $\alpha_z(z) = \alpha_w(z)$ then by the invariant those added rumors are fresh, so the new $rumors_v(z) = rumors_w(z)$ contains only fresh rumors according to z . Otherwise we are safe since the assumption $\alpha_v(z) = \alpha_z(z)$ in the invariant is not satisfied, where we mean a new $\alpha_v(z) = \alpha_w(z)$.

Case 3: process v adds the current value of register of w' to all lists $rumors_v(z)$, for each z .

By a definition of freshness, this rumor is fresh according to any process z . To see this observe that it might be non-fresh only if process z had been activated during iteration $\tau + 1$ between read and write operations of process v , but then z changed its $\alpha_z(z)$ tag. Such cases are irrelevant since the assumption in the invariant is not satisfied.

Hence the invariant, and by it the lemma, are proved. \square

Complexity analysis. We need to use a number of new or modified notions for the purpose of analysis of a dynamic case. For instance, a process still busy in a iteration is simply called *active*, to distinguish it from the processes that are already pausing in the given iteration. We restrict our attention to active processes only. Define a rumor to be *fresh*, according to active process v , if it has been read from the respective register of process v after the last activation of process v . We call an epoch *a-successful* if there is a set A of processes of cardinality at least a such that for every $v \in A$ and any process z , all the rumors in the list $rumors_v(z)$ are fresh according to z .

The main results of this section are as follows.

THEOREM 4. *Algorithm D-ReCollect has collective latency of $\mathcal{O}(n \log^7 n)$.*

COROLLARY 1. *Algorithm D-ReCollect has competitive latency of $\mathcal{O}(\log^7 n)$.*

PROOF. It follows from [3] that the competitive latency is at most the collective latency divided by n and plus 1, which makes Theorem 4 directly applicable. \square

7. DISCUSSION

This paper presents a new deterministic algorithm for the *Collect* problem. The work upper bound of the algorithm substantially improves on the best previously known results. Specifically, our algorithm is the first deterministic algorithm with the *worst-case* work of $\mathcal{O}(n \log^7 n)$ that exceeds the required linear work by only a polylogarithmic factor, significantly improving on the best prior deterministic solution of Ajtai, Aspnes, Dwork, and Waarts [3].

Our solution is parameterized by graphs embedded in the code of the algorithm. The specific graphs used here are not constructive, in the formal sense that time required to compute such graphs is polynomial in the maximum degree and $\log n$. This is because our goal was to minimize the work complexity by means of the small exponent in the power of $\log n$.

To obtain a constructive solution, one can use the family of a -expanders in described by Ta-Shma, Umans, and Zuckerman [41], yielding an algorithm with worst-case work of $\mathcal{O}(n \text{polylog } n)$. The polylogarithmic factor occurring in such a bound can be obtained by taking the polylogarithmic factor, by which a -expanders given in [41] miss the lower bound n/a for the maximum degree, and dividing it by $\log n$. For completeness, we mention that the degree of a -expanding graphs in [41] is $\mathcal{O}(\frac{n}{a} \log^3 n)$, and hence the additional factor in the work bound of a polynomially-constructible collect algorithm is $\mathcal{O}(\log^2 n)$, resulting in work $\mathcal{O}(n \log^9 n)$. A similar conversion applies for a polylogarithmic competitive latency bound of the online algorithm. The analysis in [41] shows that their construction of a -expanders is polynomial in n . We use a family of such expanders, of a logarithmic size, for values of a being consecutive powers of 2. Given the graphs as in [41], an additional time to build the neighborhoods of graphs and sequences, as needed in the *Collect* solution, is $\mathcal{O}(n \text{polylog } n)$.

En route to obtaining our solution, we introduced a gossiping game on graphs and analyzed it in terms of expansion of the underlying graph. A corollary shows that *all* expanders are fault-tolerant, in the sense captured by Theorem 2, generalizing the result of Upfal [42] obtained for specific graphs.

8. REFERENCES

- [1] Y. Afek, H. Attiya, D. Dolev, E. Gafni, M. Merritt, and N. Shavit, Atomic snapshots of shared memory, *J. ACM*, 40 (1993) 873–890.
- [2] Y. Afek, G. Stupp, and D. Touitou, Long-lived and adaptive collect with applications, *Proc., 40th IEEE Symp. on Found. of Comp. Science*, 1999, pp. 262–272.
- [3] M. Ajtai, J. Aspnes, C. Dwork, and O. Waarts, A theory of competitive analysis of distributed algorithms, *Proc., 35th IEEE Symp. of Foundations of Computer Science*, 1994, pp. 401–411.
- [4] J. Anderson, Composite registers, *Distributed Computing*, 6 (1993) 141–154.
- [5] R.J. Anderson, and H. Woll, Algorithms for the certified write-all problem, *SIAM J. Computing*, 26 (1997) 1277–1283.

- [6] J. Aspnes, and M. Herlihy, Wait-free data structures in the asynchronous PRAM model, in *Proc., 2nd ACM Symp. on Parallel Alg. and Arch.*, 1990, pp. 340–349.
- [7] J. Aspnes, and W. Hurwood, Spreading rumors rapidly despite an adversary, *J. Algorithms*, 26 (1998) 386–411.
- [8] H. Attiya, A. Fouren, and E. Gafni, An adaptive collect algorithm with applications, *Distributed Computing*, 15 (2002) 87–96.
- [9] H. Attiya, M. Herlihy, and O. Rachman, Atomic snapshots using lattice agreement, *Distributed Computing*, 8 (1995) 121–132.
- [10] H. Attiya, and O. Rachman, Atomic snapshots in $\mathcal{O}(n \log n)$ operations, *SIAM J. Computing*, 27 (1998) 319–340.
- [11] B. Awerbuch, S. Kutten, and D. Peleg, Competitive distributed job scheduling, in *Proc., 24th ACM Symp. on Theory of Computing*, 1992, pp. 571–580.
- [12] N.T.J. Bailey, *The Mathematical Theory of Infectious Diseases and its Applications*, Charles Griffin, London, 1975.
- [13] Y. Bartal, A. Fiat, and Y. Rabani, Competitive algorithms for distributed data management, in *Proc., 24th ACM Symp. on Theory of Comp.*, 1992, pp. 39–50.
- [14] J. Buss, P.C. Kanellakis, P.L. Ragde, and A.A. Shvartsman, Parallel algorithms with processor failures and delays, *J. Algorithms*, 20 (1996) 45–86.
- [15] B.S. Chlebus, L. Gąsieniec, D.R. Kowalski, and A.A. Shvartsman, Bounding work and communication in robust cooperative computation, in *Proc., 16th Int. Symposium on Distributed Computing*, 2002, Springer LNCS 2508, pp. 295–310.
- [16] B.S. Chlebus, and D.R. Kowalski, Gossiping to reach consensus, in *Proc., 14th ACM Symp. on Parallel Algorithms and Architectures*, 2002, pp. 220–229.
- [17] 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 Arch.*, 2001, pp. 271–280.
- [18] A. Demers, D. Greene, C. Hauser, W. Irish, J. Larson, S. Shenker, H. Sturgis, D. Swineheart, and D. Terry, Epidemic algorithms for replicated database maintenance, in *Proc., 6th ACM Symp. on Principles of Distributed Computing*, 1987, pp. 1–12.
- [19] D. Dolev and N. Shavit, Bounded concurrent time-stamping, *SIAM J. Computing*, 26 (1997) 418–455.
- [20] C. Dwork, M. Herlihy, and O. Waarts, Bounded round numbers, in *Proc., 12th ACM Symp. on Principles of Distributed Computing*, 1993, pp. 53–64.
- [21] C. Dwork and O. Waarts, Simple and efficient bounded concurrent timestamping or bounded concurrent timestamp systems are comprehensible!, *Proc., 24th ACM Symp. on the Theory of Comp.*, 1992, pp. 655–666.
- [22] C. Georgiou, D.R. Kowalski, and A.A. Shvartsman, Efficient gossip and robust distributed computation, in *Proc., 17th Int. Symp. on Distributed Computing*, 2003, pp. 224–238.
- [23] C. Georgiou, A. Russell, and A.A. Shvartsman, Work-competitive scheduling for cooperative computing with dynamic groups, in *Proc., 35th ACM Symposium on Theory of Computing*, 2003, pp. 251–258.
- [24] M. Harchol-Balter, T. Leighton, and D. Lewin, Resource discovery in distributed networks, in *Proc., 18th ACM Symp. on Principles of Distributed Computing*, 1999, pp. 229–238.
- [25] A. Israeli and M. Li, Bounded time stamps *Proc., 28th IEEE Symp. Found. of Computer Science*, 1987, pp. 371–382.
- [26] P.C. Kanellakis, and A.A. Shvartsman, Efficient parallel algorithms can be made robust, *Distributed Computing*, 5 (1992) 201–217.
- [27] P.C. Kanellakis, and A.A. Shvartsman, *Fault-Tolerant Parallel Computation*, Kluwer Academic Pub., 1997.
- [28] R. Karp, C. Schindelhauer, S. Shenker, and B. Vöcking, Randomized rumor spreading, in *Proc., 41st IEEE Symp. on Foundations of Computer Science*, 2000, pp. 565–574.
- [29] D. Kempe, J. Kleinberg, and A. Demers, Spatial gossip and resource location protocols, in *Proc., 33rd ACM Symp. on Theory of Computing*, 2001, pp. 163–172.
- [30] M. Li, J. Tromp, and P.M.B. Vitányi, How to share concurrent wait-free variables, *J. ACM*, 43 (1996) 723–746.
- [31] N.A. Lynch, *Distributed Algorithms*, Morgan Kaufmann, 1996.
- [32] G. Malewicz, A work-optimal deterministic algorithm for the asynchronous certified write-all problem, in *Proc., 22nd ACM Symposium on Principles of Distributed Computing*, 2003, pp. 255–264.
- [33] C. Martel, A. Park, and R. Subramonian, Work-optimal asynchronous algorithms for shared memory parallel computers, *SIAM J. Computing*, 21 (1992) 1070–1099.
- [34] M.S. Pinsker, On the complexity of a concentrator, in *Proc., 7th Annual Teletraffic Conference*, 1973, pp. 318/1–318/4.
- [35] N. Pippenger, Sorting and selecting in rounds, *SIAM J. Computing*, 16 (1987) 1032–1038.
- [36] O. Reingold, S.P. Vadhan, and A. Wigderson, Entropy waves, the zig-zag graph product, and new constant-degree expanders and extractors, *Ann. Mathematics*, 155 (2002) 157–187.
- [37] 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*, 1991, pp. 351–362.
- [38] N. Shavit, “Concurrent Timestamping,” Ph.D. Thesis, The Hebrew University, 1989.
- [39] A.A. Shvartsman, Achieving optimal CRCW PRAM fault-tolerance, *Information Processing Letters*, 39 (1991) 59–66.
- [40] D. Sleator and R. Tarjan, Amortized efficiency of list update and paging rules, *Communications of the ACM*, 28 (1985) 202–208.
- [41] A. Ta-Shma, C. Umans, and D. Zuckerman, Loss-less condensers, unbalanced expanders, and extractors, in *Proc., 33rd ACM Symp. on Theory of Computing*, 2001, pp. 143–152.
- [42] E. Upfal, Tolerating a linear number of faults in networks of bounded degree, *Information and Computation*, 115 (1994) 312–320.
- [43] P.M.B. Vitányi and B. Awerbuch, Atomic shared register access by asynchronous hardware, in *Proc., 27th Symp. on Foundations of Computer Science*, 1986, pp. 233–243.
- [44] R. van Renesse, Y. Minsky, and M. Hayden, A gossip-style failure detection service, in *Proc., IFIP Int. Conf. on Distributed Systems Platforms and Open Distributed Processing*, 1998, pp. 55–70.
- [45] A. Wigderson, and D. Zuckerman, Expanders that beat the eigenvalue bound: explicit construction and applications, *Combinatorica*, 19 (1999) 125–138.