SIAM J. DISCRETE MATH.
Vol. 14, No. 2, pp. 207–222

THE WAKEUP PROBLEM IN SYNCHRONOUS BROADCAST SYSTEMS[*]

LESZEK GĄSIENIEC[†], ANDRZEJ PELC[‡], AND DAVID PELEG[§]

Abstract. This paper studies the differences between two levels of synchronization in a distributed broadcast system (or a multiple-access channel). In the *globally synchronous* model, all processors have access to a global clock. In the *locally synchronous* model, processors have local clocks ticking at the same rate, but each clock starts individually when the processor wakes up.

We consider the fundamental problem of waking up all n processors of a completely connected broadcast system. Some processors wake up spontaneously, while others have to be woken up. Only awake processors can send messages; a sleeping processor is woken up upon hearing a message. The processors hear a message in a given round if and only if exactly one processor sends a message in that round. Our goal is to wake up all processors as fast as possible in the worst case, assuming an adversary controls which processors wake up and when. We analyze the problem in both the globally synchronous and locally synchronous models with or without the assumption that n is known to the processors. We propose randomized and deterministic algorithms for the problem, as well as lower bounds in some of the cases. These bounds establish a gap between the globally synchronous and locally synchronous models.

Key words. broadcast, clock, synchronous, wakeup

AMS subject classifications. 68W15, 68W20

PII. S0895480100376022

1. Introduction.

1.1. The problem.

This paper focuses on the effects of the level of synchronization required in *broadcast* systems (or *multiple-access channels*) such as the Ethernet. The communication system is assumed to be synchronous, namely, processors send messages in rounds. As the communication channel is shared by all processors, messages might collide. It is assumed that the processors succeed in hearing a message in a given round if and only if exactly one processor sends a message in that round; if more than one processor, or none of them, sends a message in that round, then nobody hears anything. Hence the communication model is equivalent to the radio model; cf. [1, 2, 5, 6, 7, 8, 9, 10, 14, 18, 19, 20, 22] in a complete graph without collision detection. As pointed out in [3], which studied the relationships between radio networks with and without collision detection, the absence of collision detection characterizes noisy networks since the noise does not allow processors to distinguish no transmission from multitransmission.

[*]Received by the editors August 1, 2000; accepted for publication (in revised form) January 19, 2001; published electronically April 3, 2001. A preliminary version of this paper appeared in *Proceedings of the* 19*th Annual ACM Symposium on Principles of Distributed Computing*, Portland, OR, 2000, pp. 113–121.

http://www.siam.org/journals/sidma/14-2/37602.html

[†]Department of Computer Science, The University of Liverpool, Liverpool L69 7ZF, United Kingdom (leszek@csc.liv.ac.uk).

[‡]Département d'Informatique, Université du Québec à Hull, Hull, Québec J8X 3X7, Canada (Andrzej_Pelc@uqah.uquebec.ca). The research of this author was supported in part by NSERC grant OGP 0008136. This research was done during the author's stay at the University of Liverpool.

[§]Department of Applied Mathematics and Computer Science, The Weizmann Institute, Rehovot 76100, Israel (peleg@wisdom.weizmann.ac.il). The research of this author was supported in part by grants from the Israel Ministry of Science and Art.

Evidently, the possibility of collisions makes broadcast systems harder to coordinate and control than standard point-to-point message passing systems, and performing even simple tasks poses serious difficulties. A central problem in such systems is therefore to establish a pattern of access to the shared communication media that will allow messages to go through with as little interruption as possible, i.e., avoid (or efficiently resolve) message collisions.

This problem is somewhat easier if the processors are required to be constantly attentive to the communication channel. This enables the system to use, for instance, round-robin based access protocols, as well as other schemes based on the processors being fully synchronized at all times. However, it is often desirable to allow a processor to stop being "alert" on the communication channel, whenever there is no traffic currently being transmitted on the channel, and the processor itself does not wish to send a message. From the point of view of the communication system, such a processor switches from "alert" to "sleeping," until such time as its participation is required again. Clearly, allowing processors to "sleep" entails a certain loss of synchronization, which must be regained when the processors become alert again and wish to communicate. This loss of synchronization and its effects are at the focus of the current study.

Specifically, in this paper we consider the fundamental problem of waking up all of n processors, numbered $1, \dots, n$, in a completely connected broadcast system. Some processors wake up spontaneously, in different rounds, while the others have to be woken up. Only awake processors can send messages. A sleeping processor wakes up upon hearing a message. This will happen on the first "successful" round, namely, the first round when exactly one processor sends a message.

We consider two settings of measuring time. In the first setting, termed the *globally synchronous* model, all processors have access to a global clock showing the current round number. The global clock is always available, and when a processor wakes up it can immediately see the current round number. The clock thus counts round numbers globally for all processors, starting in round 1. In the second setting, termed the *locally synchronous* model, each processor has a local clock. All local clocks tick at the same rate, one tick per round. However, no global count is available, and the local clock of processor i starts counting rounds in the round in which processor i wakes up. Moreover, in each of the above settings, we distinguish the situation when the size n of the system is known to all processors and the situation when n is unknown.

Our goal is to construct algorithms for waking up all processors as fast as possible via a multiple-access channel. We focus on the worst case, when an adversary controls which processors wake up spontaneously and when. The time complexity of the wakeup process is measured by the number of rounds elapsing from the time the first processor wakes up (spontaneously) and the time all processors are woken up (i.e., the first successful round).

During the execution, each of the processors is *active* (sends a message) in some rounds and *idle* in the others. The rounds in which a processor i is active are decided by a local protocol Π_i. The protocol Π_i can be thought of as generating a binary *activation sequence* α_i, designating the *activation times* of processor i. Specifically, if the sequence contains 1 in its tth position, i.e., $\alpha_i[t] = 1$, then processor i is required to be active on the tth round after it wakes up; conversely, $\alpha_i[t] = 0$ means i must remain silent.

The protocols Π_i used by the processors may assume any one of a number of forms. In the current paper we distinguish between the following types of algorithms. The simplest and most rigid type of protocol is what is hereafter referred to as a *fixed schedule*. It is specified by a *predefined* (and sufficiently long) activation sequence α_i for each processor i. The sequence α_i is constructed for each processor i in advance by a preprocessing algorithm Π^{pre} and is stored at its local memory. Once the processor i wakes up, it starts following its activation sequence α_i without deviation, regardless of any other parameters or inputs it may have. We refer to the set of sequences $A = \{\alpha_1, \dots, \alpha_n\}$ of the fixed schedule as the *activation set* of the system. Observe that using a fixed schedule makes the adversary rather powerful (in a manner of speaking), as it can use this knowledge in order to decide on its waking strategy.

Alternatively, the protocol Π_i may be an *online* distributed protocol, which is invoked locally by processor i upon waking up and starts generating the activation sequence α_i online. Such a protocol may be either deterministic or randomized.

Note that in the locally synchronous model there is no difference between a fixed schedule and a *deterministic* online algorithm; in both, the activation sequence α_i used by processor i is unique in all executions. This is no longer the case in the globally synchronous model, where in different executions, processor i may wake up on different rounds and may use the global round number as input to the algorithm Π_i, thus generating different sequences. Nevertheless, it is clear that even in the globally synchronous model, the adversary has complete knowledge of the activation sequences, as it controls the spontaneous wakeup time of the processors. The situation is radically different once we consider randomized protocols, as in this setting the adversary is prevented from knowing the activation sequences in advance.

1.2. Related work. Multiple-access channels, including systems such as the Aloha multiaccess system, the local area Ethernet network, multitapped buses, satellite communication systems, and packet radio networks, have been studied extensively in the literature (see [4, 23] and the references therein). Some of these models (in particular the Ethernet) assume an intermediate model of collision detection, which is not discussed here, in which the transmitting processor detects the fact that its message has collided. This feature naturally simplifies the wakeup problem.

Collision detection and resolution, as well as access management algorithms for multiple-access channels, were studied mainly in the queueing theory model, i.e., assuming a probability distribution on the arrival rate of messages at the different processors; cf. [4, 17, 16, 15]. Also, the wakeup problem was not considered as such in these contexts, although the complications that arise are similar. (The wakeup problem *has* been studied in a number of other contexts within the area of distributed systems; see, for example, [11, 13, 21], but the issues and techniques are naturally different, given the radically different communication model.)

The broadcast operation in multihop radio networks was studied in [1, 2, 5, 6, 9, 10]. The model used in those papers is based on one of two assumptions, namely, either there is a single *source* initiating the process, or all processors wake up spontaneously at time 0. Hence the starting point for the broadcast problem assumes that the wakeup problem, dealt with here, has already been solved. Nevertheless, there are strong links between the models.

Our model is closely related to certain restricted forms of concurrent write PRAM models. See, for example, [12].

To the best of our knowledge, the current paper is the first attempt to provide worst-case time bounds (against an adversary) for the wakeup problem in the synchronous setting.

1.3. Our results. We begin by showing that in the globally synchronous model, where all processors have access to a global clock, optimal deterministic wakeup time is exactly n in the worst case if the size n is known to all processors. We also show a randomized online algorithm that operates in time $O(\log n \cdot \log(1/\varepsilon))$ and succeeds in waking up the system with probability at least $1-\varepsilon$ under the same assumptions. In the case of unknown n we construct a deterministic wakeup algorithm working in worst-case time $4n$.

Under the locally synchronous model with known n, we construct a randomized online algorithm that operates in time $O(n\log(1/\varepsilon))$ and succeeds in waking up the system with probability at least $1-\varepsilon$. On the other hand, we construct a $O(n^2 \log n)$ deterministic wakeup algorithm. We also show that even when n is known, every deterministic wakeup algorithm requires worst-case time at least $(1+\varepsilon)n$ for some $\varepsilon > 0$. This establishes a gap in efficiency between the locally and globally synchronous models. Finally, still in the locally synchronous model with known n, we prove (nonconstructively) the existence of a fixed schedule with wakeup time $O(n\log^2 n)$.

Under the weakest assumptions, namely, in the locally synchronous model without the knowledge of n, we present two wakeup algorithms. The first is a randomized online algorithm which succeeds in waking up the system in time $O(n^2\log(1/\varepsilon))$ with probability at least $1-\varepsilon$; the second is a deterministic wakeup algorithm working in time $O(n^4\log^5 n)$.

2. The globally synchronous model.

In this section we consider the globally synchronous model, where a global clock is available to all processors, and every round has a global number which is known to all currently awake processors.

2.1. Known system size.

We first consider the case when the number of processors, n, is known to all of them. In this simplest case we have tight upper and lower bounds on the time required for waking up the system deterministically.

THEOREM 2.1. *If a global clock is available, the processors are labeled $\{1,\dots,n\}$ and the number n of processors is known to all of them, then there exists a deterministic online algorithm for waking up the system in time n in the worst case.*

Proof. Every processor sends a message only once after waking up in the earliest round whose number (modulo n) is equal to its label minus 1. Thus in each round at most one processor sends a message. Clearly every processor sends a message at most n rounds after waking up. ∎

THEOREM 2.2. *The worst-case minimum time to wake up an n-processor system by either a deterministic online algorithm or a fixed schedule is at least n, even if a global clock is available and the number n of processors is known to all of them.*

Proof. In order to prove the lower bound, consider an algorithm that guarantees wakeup time $k < n$ in the worst case. We show that this leads to a contradiction by presenting an adversary that wakes up a certain nonempty subset of processors in round 1 and prevents any processor from being the only sender of a message in any round until round k.

The adversary constructs a sequence $R_0,\dots,R_k$ of sets of integers as follows. Let $R_0 = \{1,\dots,n\}$. Suppose that R_j is already constructed. Let S_j be the set of those integers $i \in R_j$ for which there exists a round $r \le k$ such that i is the only processor in R_j with the following property: it sends a message in round r if it wakes up in round 1. Let $R_{j+1} = R_j \setminus S_j$.

It follows from the construction that the union of all sets S_j, $j < k$, has size at most k. Since $k < n$, it follows that R_k is nonempty. The adversary wakes up all processors from the set R_k in round 1. By definition of R_k, in any round $r \in \{1, \ldots, k\}$ either none or at least two processors send messages. $\square$

We next show that the problem can be solved by a polylogarithmic time *randomized* online algorithm.

Algorithm Repeated-Decay. The algorithm makes use of a variant of the procedure *Decay* presented in [2] for performing broadcast. The procedure assumes that all processors wake up at time 0, and some $1 \leq d \leq n$ processors contend on finding a free time slot and broadcasting their message. Each of the d contending processors repeatedly performs the following, up to a maximum of $k = 2\lceil \log n \rceil$ rounds. In each round it broadcasts a wakeup message and then flips a fair coin. If the outcome of the coinflip is 0, then it quits the procedure.

To adapt the procedure to our setting, we partition the time axis into consecutive blocks of k rounds each and repeatedly execute procedure Decay in each block of rounds. A processor waking up spontaneously on some round t must remain silent until the end of the current block and may start participating in procedure Decay only at the beginning of the next block. Hence all currently awake processors execute procedure Decay in an aligned manner.

THEOREM 2.3. *If a global clock is available and the number n of processors is known to all of them, then the randomized algorithm Repeated-*Decay *succeeds in waking up the system in time $O(\log n \cdot \log(1/\varepsilon))$ with probability at least $1 - \varepsilon$.*

Proof. It is shown in [2] that in a single invocation of procedure Decay, with probability at least 1/2, there will be a successful round in which exactly one of the contending processors will broadcast. (Intuitively, the reason can be thought of as follows. Roughly half of the contenders quit after each round. Therefore there will likely be one or two final rounds in the phase, roughly after $\log d$ rounds, in which the number of participating contenders is small, say, one or two. On those rounds, there's a good chance of success.)

Consequently, the probability that k repeated invocations of the procedure will fail to wake up the system is at most $1/2^k$. It follows that within $\log(1/\varepsilon)$ time blocks from the time the first processor woke up spontaneously, the system will be woken up with probability at least $1 - 1/2^{\log(1/\varepsilon)} = 1 - \varepsilon$. $\square$

2.2. Unknown system size. If the number of processors is not known to any of them but the global clock is available, it is still possible to wake up the system by a deterministic online algorithm in linear time.

Algorithm Interleave. For any positive integer i, partition the set of all rounds into segments $R_1^i, R_2^i, \ldots$ of length 2^i, starting from round 1, i.e., $R_j^i = \{(j-1)2^i + 1, \ldots, j \cdot 2^i\}$. Consider the following schedule. Nodes 1 and 2 send messages, respectively, in the first and second round of each segment R_j^1 of length 2. Nodes 3 and 4 send messages, respectively, in the first and second round of each odd segment of length 2, i.e., R_j^1 for odd j. Nodes 5, 6, 7, and 8 send messages, respectively, in the first, second, third, and fourth round of odd segments of length 4, i.e., R_j^2 for odd j. In general, for any $i > 0$, processors $2^i + 1, \ldots, 2^{i+1}$ send messages in consecutive rounds of odd segments of length 2^j, i.e., R_j^i for odd j.

Figure 2.1 illustrates the schedule. Note that in the set $S_j = \{i_1 < i_2 < \cdots\}$ of nodes transmitting on any given round j, the node numbers grow at least exponentially, i.e., $i_{l+1} \geq 2i_l$ for every $l \geq 1$.

THEOREM 2.4. *For an n-processor system, if a global clock is available, then the deterministic online algorithm Interleave succeeds in waking up the system in time at*

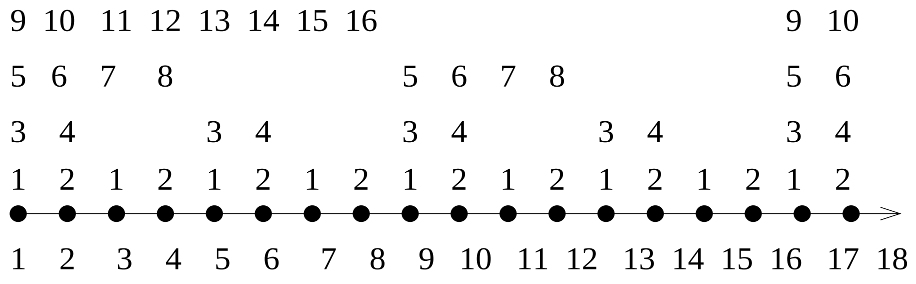

Fig. 2.1. *The schedule used by algorithm Interleave. Each node i is listed on all rounds in which it transmits.*

most $4n$, even when the number n of processors is not known to any of them.

Proof. Suppose that $2^k < n \leq 2^{k+1}$, and let r' be the round when the first processor is woken up. Let $r \leq r' + 2^{k+1}$ be the first round in which any processor sends a message. Let $S \subseteq \{1, \dots, n\}$ be the set of all processors woken up by the adversary. We will show that some processor in S is the only one to send a message in a round $j \leq r + 2^{k+1} < r' + 4n$.

Let x_1 be the largest processor sending a message in round r. If no other processor sends a message in round r, we are done. Otherwise, let $x_2 < x_1$ be the largest such processor different from x_1. We have $x_2 \leq 2^k$. Consequently, in some round $r_2 \leq r + 2^k$, x_2 is the largest processor that sends a message. If no other processor sends a message in round r_2, we are done. Otherwise, let $x_3 < x_2$ be the largest such processor different from x_2. We have $x_3 \leq 2^{k-1}$. Consequently, in some round $r_3 \leq r + 2^k + 2^{k-1}$, x_3 is the largest processor that sends a message. Using this reasoning inductively, we conclude that there is a round $j \leq r + 2^{k+1} < r' + 4n$ in which exactly one processor in S sends a message. (Indeed, if processor 1 or processor 2 is the largest one to send a message in a round, it is also unique.) □

3. The locally synchronous model with known n.

In this section we consider the locally synchronous model, where only local (equal rate) clocks are available at processors, and the local clock of each processor starts measuring time on the round when the processor wakes up. We assume that the size n of the system is known to all processors. In section 3.1 we present a randomized algorithm for the problem. We then turn to fixed schedules. Following section 3.2, which provides some necessary terminology, in section 3.3 we present a deterministic wakeup algorithm, section 3.4 describes a randomized schedule construction algorithm, and section 3.5 establishes a lower bound on wakeup time with a fixed schedule.

3.1. Randomized online algorithm.

Consider the following straightforward randomized algorithm.

Algorithm Rand-Try. Upon waking up spontaneously, each processor performs the following in each round. It randomly sets a bit

$$b \leftarrow_R \begin{cases} 1 & \text{with probability } 1/n, \\ 0 & \text{with probability } 1 - 1/n. \end{cases}$$

If the outcome is $b = 1$, then it broadcasts a wakeup message.

Theorem 3.1. *If the number n of processors is known to all of them, then the randomized algorithm Rand-Try succeeds in waking up the system in time $O(n \log(1/\varepsilon))$*

with probability at least $1-\varepsilon$.

Proof. Let $\mathcal{W}(t)$ denote the event of a successful wakeup in a round t in which k processors are awake. The success probability of this event is

$$\mathbb{P}(\mathcal{W}(t)) \;=\; \left(1-\frac{1}{n}\right)^{k-1}\cdot\frac{1}{n}\cdot k \;\geq\; \left(1-\frac{1}{n}\right)^{n}\cdot\frac{1}{n} \;\geq\; \frac{1}{2en}\;.$$

Hence the probability that none of the first $\lceil 2en\ln(1/\varepsilon)\rceil$ rounds succeed is at most

$$\mathbb{P}\left(\bigcap_{r\in R}\overline{\mathcal{W}(t)}\right) \;\leq\; \left(1-\frac{1}{2en}\right)^{2en\ln(1/\varepsilon)} \;\leq\; (1/e)^{\ln(1/\varepsilon)} \;=\; \varepsilon.$$

Hence after $O(n\log(1/\varepsilon))$ from the time the first processor woke up spontaneously, the system will be woken up with probability at least $1-\varepsilon$. ∎

3.2. Fixed schedules. Throughout the remainder of this section, we concentrate on fixed schedules (including deterministic algorithms as a special case). Let us begin by introducing some necessary terminology concerning activation sequences and executions. To begin with, note that each processor i may wake up and start its activation sequence α_i at a different time p_i (controlled by the adversary). As we start counting time from the round in which the first processor wakes up, at least one processor i must have $p_i = 0$. A sequence α_i is said to be *aligned* if it starts at time $p_i = 0$.

The resulting set of *start-time* assignments for the processors is denoted $P = \{(i, p_i) \mid 1 \leq i \leq n\}$. Given such a set P, it is possible to view each sequence α_i as shifted to the right by p_i positions, and padded by zeros at the left, thus yielding some *actual activation sequence* α_i^P which is the actual sequence governing the actions of processor i in the execution corresponding to P. Hence when we talk hereafter about bit position t of the sequence α_i under the shift pattern P, we actually look at $\alpha_i^P[t]$, the tth bit of the sequence α_i^P, or equivalently at $\alpha_i[t-p_i]$, the $(t-p_i)$th bit of the original sequence α_i (if $t < p_i$, then this bit is zero by default).

The set P is henceforth referred to as the *shift pattern* of the execution. Later on we will also consider *partial* shift patterns, specifying the start-times for only some of the activation sequences, i.e., $P = \{(i_j, p_{i_j}) \mid 1 \leq j \leq k\}$ for some $k < n$. For an activation set A and a shift pattern P, the set of shifted activation sequences resulting from applying P to A is denoted by $A^P = \{\alpha_i^P \mid \alpha_i \in A\}$. Recall that every shifted activation set A^P must contain at least one aligned sequence, as every shift pattern P contains at least one pair (i, p_i) with $p_i = 0$. The original activation set (or equivalently, the set A^P resulting from selecting the starting time $p_i = 0$ for all sequences) is referred to as the *aligned* activation set.

DEFINITION 3.2. *The bit position $t \geq 0$ is* covered *by the shifted activation set A^P if there is exactly one sequence $\alpha_i \in A$ satisfying $\alpha_i^P[t] = 1$, and the rest have $\alpha_j^P[t] = 0$. Position t is* blocked *by A^P if it is not covered by it. It is* filled *by A^P if there is* at least *one sequence $\alpha_i \in A$ such that $\alpha_i^P[t] = 1$.*

Two sequences α_i^P and α_j^P in the shifted activation set A^P collide *in position t if $\alpha_i^P[t] = \alpha_j^P[t] = 1$.*

For integers $k \geq m \geq 0$, the shift pattern P is said to be $[m,k]$-blocking *for A if the shifted activation set A^P blocks every bit position $m \leq t \leq k$.*

For an activation set A, let $W(A)$ denote the worst-case wakeup time of A, namely, the minimal integer k such that there does not exist a $[0,k]$-blocking shift pattern P for A.

Note that an algorithm for the wakeup problem does not need to generate activation sequences of infinite length. Assuming it generates the sequences of an activation set A with $W(A) < \infty$, one bit at a time, it suffices to generate the first $W(A)$ bits.

Since the adversary controls the times when the processors wake up (i.e., it controls the shift pattern P), our problem of minimizing the wakeup time of the system is equivalent to constructing an activation set A of n binary sequences α_i of minimal wakeup time $W(A)$. Without loss of generality, all sequences start with bit 1, i.e., $\alpha_i[1] = 1$ for all $i = 1, \ldots, n$. (This is because there is a one-to-one correspondence between sequences of leading zeros and late wakeup times, and the latter are under the control of the adversary. Hence $W(A)$ cannot be improved by, say, padding each sequence $\alpha_i \in A$ with z_i leading zeros, since the adversary can always nullify the effect of the leading zeros, and mimic the worst shift pattern P for A on the modified activation set, simply by making every processor i wake up z_i rounds earlier than in P.)

3.3. A deterministic online algorithm.

Algorithm Prime-Steps. Let q_i, for $i = 1, \ldots, n$, be the ith prime number larger than n. We define a set of sequences $A = \{\alpha_i : i = 1, \ldots, n\}$ of lengths $m_i = nq_i + 1$ as follows. The sequence α_i, describing the behavior of processor i, has bit 1 on positions kq_i, for natural $k = 0, 1, \ldots, n$, and bit 0 on all other positions.

For example, suppose that $n = 4$. Then we take $q_1 = 5$, $q_2 = 7$, $q_3 = 11$, and $q_4 = 13$, and hence the (periodic) sequences of rounds $\bar{t}_i$ in which node i transmits (after waking up) are

$$\begin{aligned}\bar{t}_1 &= (0, 5, 10, 15, 20),\\ \bar{t}_2 &= (0, 7, 14, 21, 28),\\ \bar{t}_3 &= (0, 11, 22, 33, 44),\\ \bar{t}_4 &= (0, 13, 26, 39, 52).\end{aligned}$$

The choice of algorithm Prime-Steps ensures that the sequences assigned to different processors collide rather infrequently, as detailed in the proof of the following theorem.

THEOREM 3.3. *Algorithm Prime-Steps wakes up a system of n processors in time $O(n^2 \log n)$.*

Proof. Since q_i is of size $O(n \log n)$, for all $i = 1, 2, \ldots, n$, we have $m_i \in O(n^2 \log n)$. Fix a shift pattern P. It suffices to show that there exists a bit position covered by A^P. Let i be the processor that wakes up first, i.e., $p_i = 0$. If bit position 0 is not covered, then some other sequence $\alpha^P_{j_1}$ necessarily has 1 in position 0. It follows that sequences α^P_i and $\alpha^P_{j_1}$ do not collide in positions $t > 0$. (Since q_i and q_{j_1} are primes, α^P_i and $\alpha^P_{j_1}$ collide as rarely as $q_i q_{j_1}$ which is larger than m_i and m_{j_1}.) If bit position q_i is not covered, it means that some sequence $\alpha^P_{j_2}$, different from α^P_i and $\alpha^P_{j_1}$, has 1 in position q_i. Again, sequences α^P_i and $\alpha^P_{j_2}$ do not collide in positions $t > q_i$. Consequently, in order to guarantee that bit positions kq_i, for $i = 0, 1, 2, \ldots$ are not covered, the adversary must generate a collision with a different sequence α_{j_k} for each bit position kq_i. Since there are only $n - 1$ sequences different from α_i, it follows that one of the bit positions kq_i, for $i = 0, 1, \ldots, n-1$, must be covered. Since $nq_i < m_i$, the theorem follows. ∎

3.4. The existence of a short fixed schedule. We now show that there exists a fixed schedule guaranteeing much faster wakeup times than those given by the deterministic online algorithm Prime-Steps. The proof is nonconstructive; we describe a randomized (Monte-Carlo) preprocessing algorithm Π^{random}, which randomly selects an activation set A for n processors, with the property that $W(A) = O(n \log^2 n)$ with probability at least $1 - \frac{1}{n}$. This implies that there must *exist* a fixed schedule with wakeup time $O(n \log^2 n)$, as the nonexistence of such a schedule would force the success probability of algorithm Π^{random} to be zero. However, we know of no efficient (deterministic or randomized) algorithmic way for ascertaining the wakeup time $W(A)$ of any given activation set A, whether constructed by the preprocessing algorithm Π^{random} or produced by any other means. Subsequently, we do not have a way of transforming algorithm Π^{random} into a polynomial expected time Las Vegas algorithm for constructing fixed schedules.

Set[1] $m = cn \log n \ln n$, for $c = 33$.

Preprocessing algorithm Π^{random}. Construct the sequences α_i of the activation set A by randomly setting each bit $\alpha_i[t]$, for $1 \le i \le n$ and $0 \le t \le m$, fixing

$$\alpha_i[t] \leftarrow_R \begin{cases} 1 & \text{with probability } 1/n, \\ 0 & \text{with probability } 1 - 1/n. \end{cases}$$

Analysis. For every $0 \le \ell \le \log n$, let $T_\ell = cn \ln n \cdot \ell$.

For every $0 \le t \le m$, let $w(t)$ denote the number of processors already awake by time t, $w(t) = |\{i \mid p_i \le t\}|$. Note that $w(t)$ is nondecreasing in the range $0 \le t \le m$.

DEFINITION 3.4. *P is said to be* ℓ-regular *if $w(T_\ell) \ge 2^\ell$ and $w(T_{\ell+1}) < 2^{\ell+1}$. Let $I(P)$ denote the smallest index such that P is* ℓ-regular,

$$I(P) \;=\; \min\{\ell \mid P \text{ is } \ell\text{-regular}\} \;.$$

Note. For every P, there exists some $0 \le \ell \le \log n$ such that P is ℓ-regular; hence $0 \le I(P) \le \log n$.

Partition the shift patterns into classes $\mathcal{P}_0, \ldots, \mathcal{P}_{\log n}$, such that $\mathcal{P}_\ell$ contains all shift patterns P for which $I(P) = \ell$,

$$\mathcal{P}_\ell = \{P \mid I(P) = \ell\} \;.$$

Consider a shift pattern $P \in \mathcal{P}_\ell$. Let $Cov(P,t)$ be the event that bit position t for $T_\ell \le t \le m$ is covered by P (i.e., it has exactly one 1 and $w(t) - 1$ zeros). Then

$$\begin{aligned}\mathbb{P}(Cov(P,t)) &\ge w(t) \cdot \frac{1}{n} \cdot \left(1 - \frac{1}{n}\right)^{w(t)-1} \\ &\ge 2^\ell \cdot \frac{1}{n} \cdot \left(1 - \frac{1}{n}\right)^{n} \;\approx\; \frac{2^\ell}{en} \;.\end{aligned}$$

Let $\mathcal{B}(P,\ell)$ be the event that P is $[T_\ell, T_{\ell+1}]$-blocking. Then

$$\begin{aligned}\mathbb{P}(\mathcal{B}(P,\ell)) &\le \left(1 - \frac{2^\ell}{en}\right)^{cn \ln n} \;\approx\; \exp(-c \ln n \cdot 2^\ell / e) \\ &\le \exp(-c' \ln n \cdot 2^\ell) \;, \qquad (1)\end{aligned}$$

[1] For notational simplicity we ignore rounding throughout the paper; whenever an integer value is called for, $\log n$ and $\ln n$ should be thought of as rounded upwards to $\lceil \log n \rceil$ and $\lceil \ln n \rceil$, respectively.

for $c' = 12$.

For any two sets of processors S_1, S_2 let $\mathcal{P}_\ell[S_1, S_2]$ denote the subclass of $\mathcal{P}_\ell$ in which

$$\begin{cases} 0 \le p_j \le T_\ell, & j \in S_1 \ , \\ T_\ell < p_j \le T_{\ell+1}, & j \in S_2 \ , \\ T_{\ell+1} < p_j, & j \notin S_1 \cup S_2 \ . \end{cases}$$

Note that the class $\mathcal{P}_\ell$ is the union of subclasses $\mathcal{P}_\ell[S_1, S_2]$ over all possible choices of processor sets S_1, S_2 such that $2^\ell \le |S_1| < 2^{\ell+1}$ and $0 \le |S_2| < 2^{\ell+1} - |S_1|$. Formally, letting

$$\mathcal{S} \ = \ \{(S_1, S_2) \mid 2^\ell \le |S_1| < 2^{\ell+1} \text{ and } 0 \le |S_2| < 2^{\ell+1} - |S_1|\} \ ,$$

we have

$$\mathcal{P}_\ell \ = \bigcup_{(S_1,S_2)\in\mathcal{S}} \mathcal{P}_\ell[S_1, S_2] \ .$$

Let $\beta(S_1)$ denote the set of all shift patterns for the processors of S_1 in the range $[0, T_\ell]$. Thus, for $S_1 = \{j_1, \ldots, j_s\}$,

$$\beta(S_1) \ = \ \{(p_{j_1}, \ldots, p_{j_s}) \mid 0 \le p_{j_l} \le T_\ell \text{ for } 1 \le l \le s\} \ .$$

Similarly, let $\gamma(S_2)$ denote the set of all shift patterns for the processors of S_2 in the range $(T_\ell, T_{\ell+1}]$, i.e., for $S_2 = \{j_1, \ldots, j_r\}$,

$$\gamma(S_2) \ = \ \{(p_{j_1}, \ldots, p_{j_r}) \mid T_\ell < p_{j_l} \le T_{\ell+1} \text{ for } 1 \le l \le r\} \ .$$

For shift patterns $\beta \in \beta(S_1)$ and $\gamma \in \gamma(S_2)$, let $\mathcal{P}_\ell[\beta, S_1, \gamma, S_2]$ denote the subclass of $\mathcal{P}_\ell[S_1, S_2]$ consisting of all shift patterns $P \in \mathcal{P}_\ell$ that match β over S_1 and γ over S_2. Hence

$$\mathcal{P}_\ell \ = \bigcup_{(S_1,S_2)\in\mathcal{S}} \ \bigcup_{\substack{\beta\in\beta(S_1)\\ \gamma\in\gamma(S_2)}} \mathcal{P}_\ell[\beta, S_1, \gamma, S_2] \ .$$

For any set $\mathcal{E}$ of shift patterns, let $\mathcal{B}(\mathcal{E}, \ell)$ denote the event that some $P \in \mathcal{E}$ is $[T_\ell, T_{\ell+1}]$-blocking and let $\mathcal{B}(\mathcal{E})$ denote the event that some $P \in \mathcal{E}$ is blocking. Clearly,

$$\mathbb{P}(\mathcal{B}(\mathcal{E})) \ \le \ \mathbb{P}(\mathcal{B}(\mathcal{E}, \ell)) \tag{2}$$

for every $\mathcal{E}$ and ℓ. Note that the event $\mathcal{B}(\mathcal{P}_\ell[\beta, S_1, \gamma, S_2], \ell)$ happens if and only if *every* $P \in \mathcal{P}_\ell[\beta, S_1, \gamma, S_2]$ is $[T_\ell, T_{\ell+1}]$-blocking, since for all $P, P' \in \mathcal{P}_\ell[\beta, S_1, \gamma, S_2]$, P and P' have the same shift configuration in the range $(T_\ell, T_{\ell+1}]$; hence P is $[T_\ell, T_{\ell+1}]$-blocking if and only if P' is $[T_\ell, T_{\ell+1}]$-blocking. Hence for every $P \in \mathcal{P}_\ell[\beta, S_1, \gamma, S_2]$,

$$\mathbb{P}(\mathcal{B}(\mathcal{P}_\ell[\beta, S_1, \gamma, S_2], \ell)) \ = \ \mathbb{P}(\mathcal{B}(P, \ell)) \ ,$$

and using inequalities (2) and (1),

$$\begin{aligned} \mathbb{P}(\mathcal{B}(\mathcal{P}_\ell[\beta, S_1, \gamma, S_2])) &\le \mathbb{P}(\mathcal{B}(\mathcal{P}_\ell[\beta, S_1, \gamma, S_2], \ell)) \\ &\le \exp(-c' \ln n \cdot 2^\ell) \ . \end{aligned} \tag{3}$$

As

$$\mathcal{B}(\mathcal{P}_\ell) \;=\; \bigcup_{(S_1,S_2)\in\mathcal{S}} \;\; \bigcup_{\substack{\beta\in\beta(S_1)\\ \gamma\in\gamma(S_2)}} \mathcal{B}(\mathcal{P}_\ell[\beta,S_1,\gamma,S_2]) \;,$$

we have that

$$\begin{aligned}\mathbb{P}(\mathcal{B}(\mathcal{P}_\ell)) &= \sum_{(S_1,S_2)\in\mathcal{S}} \;\; \sum_{\substack{\beta\in\beta(S_1)\\ \gamma\in\gamma(S_2)}} \mathbb{P}(\mathcal{B}(\mathcal{P}_\ell[\beta,S_1,\gamma,S_2]))\\ &\le \sum_{(S_1,S_2)\in\mathcal{S}} |\beta(S_1)|\cdot|\gamma(S_2)|\cdot\mathbb{P}(\mathcal{B}(P,\ell)) \;.\end{aligned}$$

As

$$\begin{aligned}|\beta(S_1)| = (T_\ell+1)^s \;&=\; (1+cn\ln n\cdot\ell)^s \;,\\ |\gamma(S_2)| = (T_{\ell+1}-T_\ell)^r \;&=\; (cn\ln n)^r \;\le\; (cn\ln n)^{2^{\ell+1}-s} \;,\end{aligned}$$

we have

$$|\beta(S_1)|\cdot|\gamma(S_2)| \;\le\; (1+cn\ln n\log n)^{2^{\ell+1}} \;;$$

hence by inequality (3)

$$\begin{aligned}\mathbb{P}(\mathcal{B}(\mathcal{P}_\ell)) &\le \sum_{(S_1,S_2)\in\mathcal{S}} (1+cn\ln n\log n)^{2^{\ell+1}}\cdot\exp(-c'\ln n\cdot 2^\ell)\\ &\le \sum_{s\le 2^{\ell+1}}\binom{n}{s}\sum_{r\le 2^{\ell+1}-s}\binom{n-s}{r}\cdot\exp(2\ln n\cdot 2^{\ell+1}-c'\ln n\cdot 2^\ell)\\ &\le n^{2^{\ell+2}+2}\cdot\exp((4-c')\ln n\cdot 2^\ell)\\ &\le \exp((10-c')\ln n\cdot 2^\ell)\\ &\le \exp(-2\ln n) \;.\end{aligned}$$

$\mathcal{B}(\mathcal{P}) = \bigcup_\ell \mathcal{B}(\mathcal{P}_\ell)$, so

$$\mathbb{P}(\mathcal{B}(\mathcal{P})) \;\le\; \log n\cdot\mathbb{P}(\mathcal{B}(\mathcal{P}_\ell)) \;\le\; \frac{\log n}{n^2} \ll \frac{1}{n} \;.$$

The above analysis yields the following theorem.

THEOREM 3.5. *For an n-processor system with n known to the processors, algorithm Π^{random} constructs an activation set A whose wakeup time is $W(A) = O(n\log^2 n)$ with probability at least $1-\frac{1}{n}$.*

COROLLARY 3.6. *For an n-processor system with n known to the processors, there exists a fixed schedule with wakeup time $O(n\log^2 n)$.*

It is important to underline the difference between algorithm Rand-Try and algorithm Π^{random}. Although randomness is involved in both of them, the former is an online algorithm, while the latter is a randomized method to produce a *fixed schedule*. Consequently, in algorithm Rand-Try the adversary does not know the behavior of processors in advance and must decide if and when to wake up each of them based only on the history to date. In the second scenario, on the other hand, the adversary is given a fixed schedule in advance, and hence it also has knowledge of the behavior of processors in the future. This additional information gives the adversary more power which is reflected by the worse performance of the fixed schedule, as compared to the randomized online algorithm Rand-Try. In principle, our existence proof can be used for *generating* a (provably correct) short schedule; simply construct schedules one by one as indicated by the proof and test each of them until finding one satisfying the requirements. Unfortunately, testing the correctness of a given schedule (i.e., verifying that it succeeds in waking up the processors against any adversarial behavior) seems to be a difficult task, and we do not see any way of achieving it short of the naive brute-force testing of all possible schedules.

3.5. Lower bound for fixed schedules. We do not know what is the optimal time of a deterministic online wakeup algorithm (or fixed schedule). Below we give a lower bound that serves primarily to establish a gap between the global clock and local clocks scenarios; while in the former scenario we showed an algorithm working in time n, it turns out that without a global clock, the wakeup time of any fixed schedule (including in particular those generated by a deterministic algorithm) must be greater than $(1+\varepsilon)n$ for some positive constant ε.

Consider an activation set consisting of n infinite binary activation sequences, $A = \{\alpha_1, \dots, \alpha_n\}$. We show that the adversary can select the waking times of processors, i.e., the shift pattern P for these sequences on the time axis, in such a way that no message is heard within the first $(1+\varepsilon)n$ rounds, counted from waking up the first processor, for some constant $\varepsilon > 0$. (No attempt has been made to optimize the constant. We give the proof for $\varepsilon = 0.001$, when n is sufficiently large.)

To establish the lower bound, it suffices to show that for some $1 \le x \le n$, there exist an activation set $B \subset A$ of cardinality x and an $[0, x+\varepsilon n]$-blocking shift pattern P for B. The remaining $n-x$ sequences can be used to block at least $n-x$ additional positions, $x+\varepsilon n+1, \dots, n+\varepsilon n$, using one shifted sequence to block each of these positions.

Lemma 3.7 (blocking technique). *Let Q be any set of m (not necessarily consecutive) positions and let B be any aligned activation set of $m+1$ sequences. Then there exists a nonempty subset of B which blocks all positions in Q.*

Proof. Repeat the following process at most m times. For any position j in Q covered by B, remove the sequence covering j from the set B. After removing at most m sequences, all positions of Q are blocked and at least one sequence remains in B. ∎

Let δ and σ be positive integer constants to be determined later and let $N = n/\sigma$.

For any sequence α_i, the *m-head* of α_i, denoted $head(\alpha_i, m)$, is its prefix of length m. Let $ones(\alpha_i, m)$ denote the number of ones in $head(\alpha_i, m)$.

Lemma 3.8. *For every two integers $a \ge 1$ and $m \ge 2^a$, and for every aligned activation set B of m or more sequences satisfying $ones(\alpha_i, m) \le a$, there exist an integer $0 \le \ell \le a-1$ and a set $C \subseteq B$ of size $|C| > m/2^{\ell+1}$ such that every $\alpha_i \in C$ contains zeros at all positions $m/2^{\ell+1}+1$ to $m/2^{\ell}$.*

Proof. The proof is by induction on a. For $a = 1$, each sequence in B contains a single 1 at the first position and zeros at all other positions; hence the set $C = B$ satisfies the claim with $\ell = 0$. Considering $a > 1$, assume the claim holds for every $a' < a$. Let B' be the set of all sequences $\alpha_i \in B$ which contain a 1 at some position from $m/2+1$ to m. If $|B'| < m/2$, then the claim holds for $\ell = 0$ and $C = B \setminus B'$, and we are done. Otherwise, letting $m' = m/2$ and $a' = a-1$, the set B' satisfies $|B'| \ge m'$ and every sequence $\alpha_i \in B'$ satisfies $ones(\alpha_i, m') \le a'$. Hence by the inductive hypothesis, there exist an integer $0 \le \ell' \le a'-1$ and a set $C \subseteq B'$ of size

$|C| > m'/2^{\ell'+1}$ such that every $\alpha_i \in C$ contains zeros at all positions $m'/2^{\ell'+1} + 1$ to $m'/2^{\ell'}$. The claim now holds for C and $\ell = \ell' + 1$. $\square$

Returning to the analysis, there are two cases.

Case 1*:* At least N sequences have N-heads with at most $\delta - 1$ ones.

Let B be an activation set consisting of N sequences with the above property. By Lemma 3.8, there exist an integer $0 \le \ell \le \delta - 2$ and a set $C \subseteq B$ of size $|C| > N/2^{\ell+1}$ such that every $\alpha_i \in C$ contains zeros at all positions $N/2^{\ell+1} + 1$ to $N/2^{\ell}$. Using the blocking technique of Lemma 3.7, we can now use a subset $C' \subseteq C$ of at most $N/2^{\ell+1} + 1$ sequences to block the first $N/2^{\ell+1}$ positions and gain $N/2^{\ell+1}$ blocked positions "for free." Then we use the remaining $n - |C'|$ sequences to block $n - |C'|$ additional positions for a total of $n + \frac{N}{2^{\ell+1}} - 1$ positions. In the worst case, occurring when $\ell = \delta - 2$, this amounts to $n + \frac{n}{\sigma \cdot 2^{\delta-1}} - 1$ blocked positions.

Case 2*:* More than $n - N$ sequences have N-heads with at least δ ones.

Let $M = n - N$ and let B be an activation set of M sequences with the above property. Let I_n denote the time interval of the first $n + 1$ positions, $0, 1, \ldots, n$. We show that we can block all positions of I_n by selecting a subset $C \subseteq B$ of ρn sequences, for $\rho \le 1 - 1/\sigma$, and a shift pattern P for C, in such a way that each position of I_n is filled by at least two sequences of C^P. Assume, without loss of generality, that the number of ones in each N-head is exactly δ (ignore other ones).

For each sequence $\alpha_i \in B$ we consider only M possibilities of right shifts from the aligned position, namely, by $p_i \in \{1, \ldots, M\}$. We call these the *possible* shifts. For each of them, the entire N-head of α_i corresponds to positions within the interval I_n. We construct a shift pattern P for a subset of sequences $C \subseteq B$. The set C is initialized to $\emptyset$. Our goal is to fill all positions in I_n by C^P. Suppose that at some stage of the construction, some positions in I_n are already filled by C^P. Selecting a new sequence α_i to be added to C, we look for a shift p_i that fills as many new positions as possible (i.e., we try to align many ones from the N-head of α_i with yet unfilled positions in I_n).

LEMMA 3.9. *If the number of filled positions in I_n is strictly less than kM/δ, for $1 \le k \le \delta$, then for any sequence $\alpha_i \in B$ there exists a possible shift p_i which fills at least $\delta - k + 1$ new positions in I_n.*

Proof. Assume the contrary and consider a sequence α_i such that for each possible shift p_i of α_i, at least k ones of its N-head correspond to filled positions in I_n. Thus in total we have at least kM such *matches*.

On the other hand, each filled position in I_n can be matched with at most δ ones from the N-head of α_i. There are strictly fewer than kM/δ filled positions in I_n, so the total number of matches is strictly less than kM, a contradiction. $\square$

The process of constructing C and filling the positions of the interval I_n is divided into δ consecutive stages. In stage k, when the number of filled positions in I_n is at least $(k-1)M/\delta$ but strictly smaller than kM/δ, it is possible to add in each step an appropriately shifted sequence to C filling at least new $\delta - k + 1$ positions in I_n. Thus stage k consists of at most $(\frac{kM}{\delta} - \frac{(k-1)M}{\delta}) \cdot \frac{1}{\delta-k+1}$ such steps, and the total number of sequences needed to fill all positions of I_n is at most

$$\sum_{k=1}^{\delta} \left\lceil \frac{1}{\delta - k + 1} \cdot \frac{M}{\delta} \right\rceil + N \;\le\; \frac{M}{\delta} \sum_{k=1}^{\delta} \frac{1}{k} + N + \delta.$$

Notice that each of the last N sequences is used to fill only a single position thus we can use sequences outside of B for this purpose.

It follows that all positions of I_n can be *blocked* by repeating this process twice, and the number of sequences used is at most

$$2n\left(\frac{1}{\delta}\left(1-\frac{1}{\sigma}\right)\sum_{k=1}^{\delta}\frac{1}{k}+\frac{1}{\sigma}\right)+2\delta\ .$$

Finally take constants $\delta = 8$ and $\sigma = 5$. Then $\frac{1}{\delta}\sum_{k=1}^{\delta}\frac{1}{k} < 0.34$, and the total number of sequences used to block all positions of I_n is at most $\lceil 0.95n\rceil + 16$. The remaining $\lfloor 0.05n\rfloor - 16$ sequences are used to block $\lfloor 0.05n\rfloor - 16$ additional positions for a total of $\lfloor 1.05n\rfloor - 16$ positions. In Case 1 we could block at least $n + \frac{n}{\sigma\cdot 2^{\delta-1}} - 1 = n + \frac{n}{640} - 1$ positions. This gives a lower bound $1.001n$ in both cases for sufficiently large n.

THEOREM 3.10. *If a global clock is not available, then every deterministic algorithm waking up a system of n processors must use worst-case time at least $(1+\varepsilon)n$ for some positive constant ε and sufficiently large n.*

4. The locally synchronous model with unknown n.

We conclude the paper by presenting two wakeup algorithms working under the weakest scenario, namely, when a global clock is not available and the size n of the system is not known to processors. The first is a randomized online algorithm waking up the system in time $O(n^2\log(1/\varepsilon))$ with probability at least $1-\varepsilon$, and the second is a deterministic online algorithm working in worst case time $O(n^4\log^5 n)$.

4.1. Randomized algorithm Size-Probing.

For any integer $j \geq 1$ let $T_j = \lceil e\ln(1/\varepsilon)\rceil\cdot 2^{j+1}$. Also let $S_i = \sum_{j=1}^{i} T_j = \lceil e\ln(1/\varepsilon)\rceil\cdot(2^{j+2}-2)$. Any processor, after waking up spontaneously, operates in consecutive phases numbered by positive integers. Phase j lasts T_j rounds. In each of these rounds the processor randomly sets a bit

$$b \leftarrow_R \begin{cases} 1 & \text{with probability } 1/2^j,\\ 0 & \text{with probability } 1-1/2^j.\end{cases}$$

If the outcome is $b = 1$, then it broadcasts a wakeup message.

THEOREM 4.1. *Algorithm Size-Probing succeeds in waking up an n-processor system in time $O(n^2\log(1/\varepsilon))$ with probability at least $1-\varepsilon$.*

Proof. Let i be the unique integer satisfying that $2^{i-1} < n \leq 2^i$. Our analysis focuses on what happens in the system beginning on the *special* round τ which is the first round such that some processor wakes up (spontaneously) on round τ and no processor wakes up on rounds $\tau+1,\ldots,\tau+S_i$. Clearly $\tau \leq nS_i = O(n^2\log(1/\varepsilon))$.

Consider the set of rounds $R = \{\tau+S_{i-1}+1,\ldots,\tau+S_i\}$. Let p be the processor that woke up on round τ and let X be the set of awake processors at the beginning of round τ (excluding p). In each round $t \in R$, processor p broadcasts with probability $1/2^i$ and all other awake processors from X broadcast with probabilities *at most* $1/2^i$. For a round $t \in R$, let $\mathcal{W}(t)$ denote the event that round $t \in R$ *succeeds*, namely, exactly one processor broadcasts. The probability that this happens is

$$\begin{aligned}\mathbb{P}(\mathcal{W}(t)) &\geq \frac{1}{2^i}\left(1-\frac{1}{2^i}\right)^n \;\geq\; \frac{1}{2^i}\left(1-\frac{1}{n}\right)^n\\ &\geq \frac{1}{2^i}\cdot\frac{1}{2e} \;=\; \frac{1}{2^{i+1}\cdot e}\ .\end{aligned}$$

Hence the probability that none of the rounds in R succeed is at most

$$\mathbb{P}\left(\bigcap_{t\in R}\overline{\mathcal{W}(t)}\right) \leq \left(1-\frac{1}{2^{i+1}\cdot e}\right)^{T_i}$$

$$\leq \left(1 - \frac{1}{2^{i+1} \cdot e}\right)^{2^{i+1} e \ln(1/\varepsilon)}$$
$$\leq (1/e)^{\ln(1/\varepsilon)} \;=\; \varepsilon.$$

Hence the system is woken up with probability at least $1-\varepsilon$ after $\tau+S_i = O(n^2 \log(1/\varepsilon))$ rounds. ∎

4.2. Deterministic algorithm Squared Prime-Steps. In order to present our last algorithm, define the integer sequence $S = \langle s_1, s_2, \ldots \rangle$ by setting

$$s_j = \begin{cases} 1, & j = 1, \\ 2, & j = 2, \\ s_{j-1}^2 \log^2 s_{j-1}, & j \geq 3. \end{cases}$$

For every processor i, find a value j, such that $s_{j-1} < i \leq s_j$, and construct the sequence α_i describing the behavior of processor i as in algorithm Prime-Steps assuming the size of the system is s_j. Take the resulting set of sequences α_i, padded by zeros on the right, to be the activation set of the algorithm.

THEOREM 4.2. *Algorithm Squared Prime-Steps wakes up a system of n processors in time $O(n^4 \log^5 n)$.*

Proof. Fix any shift pattern P. Consider the set $\mathcal{S}$ of processors that are active during the execution of algorithm Squared Prime-Steps. Let t be the largest processor number in $\mathcal{S}$ and let j be such that $s_{j-1} < t \leq s_j$. Let $\hat{\mathcal{S}} \subseteq \mathcal{S}$ be the set of all processors with indices larger than s_{j-1} and $\hat{A} = \{\alpha_i : i \in \hat{\mathcal{S}}\}$. Recall that $|\alpha_l| \in O(s_{j-1}^2 \log s_{j-1})$ for all $l \in \mathcal{S} \setminus \hat{\mathcal{S}}$. Therefore

$$\sum_{l \in \mathcal{S} \setminus \hat{\mathcal{S}}} |\alpha_l| \in s_{j-1} \cdot O(s_{j-1}^2 \log s_{j-1}) = O(s_{j-1}^3 \log s_{j-1}) \subseteq O(s_j^2).$$

This means that the time used by algorithm Squared Prime-Steps, before processors in $\hat{\mathcal{S}}$ become active, is in $O(s_j^2)$. In what follows we show that the system is woken up in time at most $|\alpha_i|$ after the first processor $i \in \hat{\mathcal{S}}$ become active. Recall that sequence α_i has $s_j + 1$ positions set to 1. Since any two sequences in $\hat{A}^P$ collide at most once (see proof of Theorem 3.3) and $|\hat{\mathcal{S}}| \leq s_j - s_{j-1}$, the number of positions where α_i^P is set to 1 but all other sequences in $\hat{A}^P$ are set to 0 is larger than $s_{j-1} + 1$. Notice that each sequence α_l^P, for $l \in \mathcal{S} \setminus \hat{\mathcal{S}}$, can collide in at most one position with the sequence α_i^P. This holds since $|\alpha_l| \in O(s_{j-1}^2 \log s_{j-1})$ and distances between consecutive ones in α_i are at least $s_j \geq s_{j-1}^2 \log^2 s_{j-1}$. Since $|\mathcal{S} \setminus \hat{\mathcal{S}}| \leq s_{j-1}$, the number of covered positions is larger than $s_{j-1} + 1 - s_{j-1} = 1$.

Hence the wakeup time is bounded by $|\alpha_i| + O(s_j^2) \subseteq O(s_j^2 \log s_j)$. Since $n \geq t$ and $s_j \leq t^2 \log^2 t$ we get the bound $O(n^4 \log^5 n)$ of wakeup time for a system of n processors. ∎

REFERENCES

[1] N. ALON, A. BAR-NOY, N. LINIAL, AND D. PELEG, *A lower bound for radio broadcast*, J. Comput. System Sci., 43 (1991), pp. 290–298.

[2] R. BAR-YEHUDA, O. GOLDREICH, AND A. ITAI, *On the time complexity of broadcast in multi-hop radio networks: An exponential gap between determinism and randomization*, J. Comput. System Sci., 45 (1992), pp. 104–126.

[3] R. Bar-Yehuda, O. Goldreich, and A. Itai, *Efficient emulation of single-hop radio network with collision detection on multi-hop radio network with no collision detection*, Distrib. Comput., 5 (1991), pp. 67–71.
[4] D. Bertsekas and R. Gallager, *Data Networks*, Prentice-Hall, Englewood Cliffs, NJ, 1987.
[5] I. Chlamtac and S. Kutten, *On broadcasting in radio networks – problem analysis and protocol design*, IEEE Trans. Comm., 33 (1985), pp. 1240–1246.
[6] I. Chlamtac and S. Kutten, *Tree based broadcasting in multihop radio networks*, IEEE Trans. Comput., 36 (1987), pp. 1209–1223.
[7] I. Chlamtac and S. Kutten, *A spatial reuse TDMA/FDMA for mobile multi-hop radio networks*, in Proceedings of the Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM), Washington, D.C., 1985, pp. 389–394.
[8] I. Chlamtac and O. Weinstein, *The wave expansion approach to broadcasting in multihop radio networks*, IEEE Trans. Comm., 39 (1991), pp. 426–433.
[9] B.S. Chlebus, L. Gasieniec, A. Gibbons, A. Pelc, and W. Rytter, *Deterministic broadcasting in unknown radio networks*, in Proceedings of the Eleventh Annual ACM-SIAM Symposium on Discrete Algorithms, San Francisco, 2000, pp. 861–870.
[10] K. Diks, E. Kranakis, D. Krizanc, and A. Pelc, *The impact of knowledge on broadcasting time in radio networks*, in Proceedings of the 7th Annual European Symposium on Algorithms, ESA'99, Prague, Czech Republic, 1999, Lecture Notes in Comput. Sci. 1643, Springer, Berlin, 1999, pp. 41–52.
[11] S. Even and S. Rajsbaum, *Unison, canon, and sluggish clocks in networks controlled by a synchronizer*, Math. Systems Theory, 28 (1995), pp. 421–435.
[12] F. Fich, R. Impagliazzo, B. Kapron, V. King, and M. Kutylowski, *Limits on the power of parallel random access machines with weak forms of write conflict resolution*, J. Comput. System Sci., 53 (1996), pp. 104–111.
[13] M.J. Fischer, S. Moran, S. Rudich, and G. Taubenfeld, *The wakeup problem*, SIAM J. Comput., 25 (1996), pp. 1332–1357.
[14] I. Gitman, R. M. Van Slyke, and H. Frank, *Routing in packet-switching broadcast radio networks*, IEEE Trans. Comm., 24 (1976), pp. 926–930.
[15] J. Goodman, A. G. Greenberg, N. Madras, and P. March, *On the stability of Ethernet*, in Proceedings of the 17th Symposium on Theory of Computing, Providence, RI, 1985, pp. 379–387.
[16] A. G. Greenberg, P. Flajolet, and R. E. Ladner, *Estimating the multiplicities of conflicts to speed their resolutions in multiple access channels*, J. ACM, 34 (1987), pp. 289–325.
[17] J. Hastad, T. Leighton, and B. Rogoff, *Analysis of backoff protocols for multiple access channels*, in Proceedings of the 19th ACM Symposium on Theory of Computing, 1987, pp. 241–253.
[18] R. E. Kahn, S. A. Gronemeyer, J. Burchfiel, and R. C. Kunzelman, *Advances in packet radio technology*, Proc. IEEE, 66 (1978), pp. 1468–1496.
[19] R. C. Kunzelman, *Overview of the ARPA packet radio experimental network*, in Proceedings of COMPCON, 1978, pp. 157–160.
[20] E. Kushilevitz and Y. Mansour, *Computation in noisy radio networks*, in Proceedings of the Ninth Annual ACM-SIAM Symposium on Discrete Algorithms, San Francisco, 1998, pp. 236–243.
[21] J. Mazoyer, *On optimal solutions to the firing squad synchronization problem*, Theoret. Comput. Sci., 168 (1996), pp. 367–404.
[22] N. Shacham and E. J. Craighill, *Dynamic routing for real-time transport in packet radio networks*, in Proceedings of the Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM), 1982, pp. 152–158.
[23] A. Tannenbaum, *Computer Networks*, Prentice-Hall, Englewood Cliffs, NJ, 1981.