Deterministic Logarithmic Completeness in the Distributed Sleeping Model

Leonid Barenboim
The Open University of Israel, Raanana, Israel

Tzalik Maimon
Ben-Gurion University of The Negev, Beer Sheva, Israel

Abstract

In this paper we provide a deterministic scheme for solving any decidable problem in the distributed *sleeping model*. The sleeping model [22, 9] is a generalization of the standard message-passing model, with an additional capability of network nodes to enter a sleeping state occasionally. As long as a vertex is in the awake state, it is similar to the standard message-passing setting. However, when a vertex is asleep it cannot receive or send messages in the network nor can it perform internal computations. On the other hand, sleeping rounds do not count towards *awake complexity*. Awake complexity is the main complexity measurement in this setting, which is the number of awake rounds a vertex spends during an execution. In this paper we devise algorithms with worst-case guarantees on the awake complexity.

We devise a deterministic scheme with awake complexity of $O(\log n)$ for solving any decidable problem in this model by constructing a structure we call *Distributed Layered Tree*. This structure turns out to be very powerful in the sleeping model, since it allows one to collect the entire graph information within a constant number of awake rounds. Moreover, we prove that our general technique cannot be improved in this model, by showing that the construction of distributed layered trees itself requires $\Omega(\log n)$ awake rounds. This is obtained by a reduction from message-complexity lower bounds, which is of independent interest. Furthermore, our scheme also works in the $\mathcal{CONGEST}$ setting where we are limited to messages of size at most $O(\log n)$ bits. This result is shown for a certain class of problems, which contains problems of great interest in the research of the distributed setting. Examples for problems we can solve under this limitation are leader election, computing exact number of edges and average degree.

Another result we obtain in this work is a deterministic scheme for solving any problem from a class of problems, denoted **O-LOCAL**, in $O(\log \Delta + \log^* n)$ awake rounds. This class contains various well-studied problems, such as MIS and $(\Delta + 1)$-vertex-coloring. Our main structure in this case is a tree as well, but is sharply different from a distributed layered tree. In particular, it is constructed in the local memory of each processor, rather than distributively. Nevertheless, it provides an efficient synchronization scheme for problems of the **O-LOCAL** class.

2012 ACM Subject Classification Theory of computation → Distributed algorithms

Keywords and phrases Distributed Computing, Sleeping Model, Complexity Class

Digital Object Identifier 10.4230/LIPIcs.DISC.2021.10

Related Version *Full Version*: `https://arxiv.org/pdf/2108.01963.pdf` [5]

Funding This work was supported by the Open University of Israel Research Fund.

Acknowledgements The authors are grateful to the anonymous reviewers for helpful comments.

1 Introduction

What can be computed within logarithmic complexity has been one of the most fundamental questions in distributed and parallel computing, since the initiation of the study of parallel algorithms in the eighties. Various problems were shown to belong to the NC class back then, i.e., the class of problems that can be solved in polylogarithmic time by a polynomial number of machines. This includes several fundamental problems, namely, $(\Delta + 1)$-coloring, Maximal Independent Set and Maximal Matching. All of these problems admit deterministic logarithmic parallel algorithms. In the distributed setting, however, these problems turned out to be much more challenging, if one aims at a deterministic solution. The first deterministic polylogarithmic solution was an $O(\log^7 n)$ time algorithm for the problem of Maximal Matching, obtained by Hanckowiak, Karonski and Panconesi [21]. More recently, polylogarithmic deterministic $(\Delta^{1+\epsilon})$-coloring was obtained by Barenboim and Elkin [3]. The problem of $(2\Delta - 1)$-edge-coloring was provided with an $O(\log^7 \Delta \log n)$-rounds deterministic algorithm by Fischer, Ghaffari and Kuhn [16]. Recently, a plethora of results were published in this field, with various improvements to the aforementioned algorithms. See, e.g., [4, 1, 15, 23], and references therein. In a very recent breakthrough, a wide class of problems have been solved using deterministic polylogarithmic number of rounds [26], including $(\Delta + 1)$-coloring and Maximal Independent Set. This was achieved by providing an efficient algorithm for the $(O(\log n), O(\log n))$-Network-Decomposition problem, which is complete in this class.

1.1 Our Results

In the current paper we investigate yet another distributed setting, namely, the *Sleeping Setting*. Several variants of this setting have attracted the attention of researchers recently [6, 9, 11, 14, 19, 22]. The particular setting and complexity measure we consider in this paper were introduced by Chatterjee, Gmyr and Pandurangan [9] in PODC'20. This sleeping setting is similar to the standard distributed $\mathcal{LOCAL}$ setting [24], but has an additional capability, as follows. In the sleeping setting, the vertices of the network graph can decide in each round to be in one of two states; a "sleep" state or an "awake" state. If all the vertices are awake all the time, the setting is identical to the standard $\mathcal{LOCAL}$ setting. However, the capability of entering a "sleep" state is where a vertex cannot receive or send messages in the network, neither can it perform internal computations. Consequently, such rounds do not consume the resources of that vertex, and shall not be counted towards a complexity measurement that aims at optimizing resource consumption. Indeed, in this setting a main complexity measurement takes into account only awake rounds. Specifically, the worst-case awake complexity of an algorithm in the sleeping setting is the worst-case number of rounds in which any single vertex is awake. In PODC'20, Chatterjee, Gmyr and Pandurangan [9] presented a Maximal Independent Set randomized algorithm with expected awake complexity of $O(1)$. Its high-probability awake complexity is $O(\log n)$, and its worst-case awake complexity is polylogarithmic. This work raised the following two important questions:

(1) Can MIS be solved within deterministic logarithmic awake complexity?
(2) Can additional problems be solved within such complexity?

In the current paper we answer these questions in the affirmative. But much more generally, we show that *any decidable problem* can be solved within deterministic logarithmic awake complexity in the distributed sleeping setting. Namely, a decidable problem is any computational problem that has a sequential deterministic algorithm that provides a correct solution within a finite sequential running time (as large as one wishes). Note that undecidable problems in the sequential setting are also undecidable in the different variants of distributed settings. For the purpose of solving decidable problems, we present a new structure, namely, a *Distributed Layered Tree* (DLT). We show that if one is able to compute a distributed layered tree, then any decidable problem can be solved within additional awake complexity of $O(1)$. This is because a DLT allows each vertex to obtain all the information of the input graph in a constant number of awake rounds, and then any decidable problem can be solved locally and consistently by all vertices using a sequential algorithm. We also prove that DLT itself can be solved in $O(\log n)$ awake rounds. In particular, this provides a deterministic logarithmic solution to the fundamental Broadcast problem. This improves the best previously-known awake complexity of this problem in the sleeping setting, due to Chang et al. [8], by at least a quadratic factor. We note that the broadcast algorithm of Chang et al. was devised for settings with additional requirements, i.e., it is more general than Broadcast in the sleeping setting. Nevertheless, it was still the state-of-the-art even in the sleeping model. Our improvement applies specifically to the sleeping model.

A natural question is how difficult the construction of DLT is. We answer this by proving a lower bound of $\Omega(\log n)$ awake rounds for the DLT problem. This lower bound is obtained by a simple but powerful tool of a reduction from *message complexity* lower bounds in the $\mathcal{LOCAL}$ model. With this lower bound, given that the DLT problem itself is a decidable problem, we obtain a tight deterministic bound of $\Theta(\log n)$ worst-case awake time on the class of decidable DLT-hard problems [1] in the sleeping model.

An additional direction of ours is the analysis of a class we define as the **O-LOCAL** class. This is a class of problems that can be solved using an acyclic orientation of the edges, by choosing a solution for each vertex after all vertices reachable from it have computed their solution, and as a function of these solutions. A notable example is $(\Delta + 1)$-coloring, where each color can be selected once all neighbors on outgoing edges have selected their own colors, such that the color does not conflict with any of them. Another example is MIS, where each decision is made after all outgoing neighbors have made their decisions. We show that any problem that belongs to this class has deterministic worst-case awake complexity of $O(\log \Delta + \log^* n)$.

In addition to the number of awake rounds, which is the main complexity measurement in this setting, we are also interested in optimizing the overall number of communication rounds. Since the DLT can be used to solve any decidable problem, it follows that certain such problems require $\Omega(n)$ communication rounds. (These are the global problems of the ordinary distributed setting. For example, the leader election is such a problem.) We investigate how close we can get to this lower bound with an algorithm of $\tilde{O}(n)$ rounds for the distributed layered tree problem. While our basic algorithm requires $O(n^2 \log n)$ communication rounds, a more sophisticated version requires only $O(n \log n \log^* n)$ communication rounds. This comes at a price of increasing the worst-case awake complexity, but only by a factor of $O(\log^* n)$.

1.2 The Sleeping Setting

The sleeping setting represents the need for energy-efficient algorithms in ad hoc, wireless and sensor networks. In such networks, the energy consumption depends on the amount of time a vertex is actively communicating or performing calculations. More importantly, significant energy is spent even if a node is idle, but awake. It was shown in experiments that the energy consumption in an idle state is only slightly smaller than when a node is active [12, 30]. This is in contrast to the sleeping state, in which energy consumption is decreased drastically. Thus, if a node may enter a sleeping mode to save energy during the course of an algorithm, one can significantly improve the energy consumption of the network during the execution of an algorithm.

[1] A *DLT-hard* problem is a problem whose solution provides a DLT within additional $o(\log n)$ awake rounds.

The sleeping model is a formulation of this premise, and is a generalization of the traditional $\mathcal{LOCAL}$ model. In the sleeping model, similarly to the $\mathcal{LOCAL}$ model, a communication network is represented by an n-vertex graph $G = (V, E)$, where vertices represent processors, and edges represent communication links. There is a global clock, and computation proceeds in synchronous discrete rounds. In each round a vertex can be in either of the two states, "sleep" or "awake". (While in the $\mathcal{LOCAL}$ model the vertices are only in the "awake" state.) If a vertex is in the "awake" state in a certain round, it can perform local computations as well as sending and receiving messages in that round. On the other hand, in a round of a "sleep" state, a vertex cannot send or receive messages, and messages sent to it by other vertices are lost. It also cannot perform any internal computations. A vertex decides a-priori about entering a "sleeping" state. That is, in order to enter a sleeping state in a certain round k, either the vertex decides about it in an awake round $k' < k$, or such a decision is hard-coded in the algorithm, and is known before its execution. Nodes in the "sleep" state consume almost no energy, and thus shall not be counted towards the energy efficiency analysis.

Initially, vertices know the number of vertices n, or an upper bound $\hat{n} \geq n$. The IDs of vertices are unique and belong to the set $[\hat{n}]$. Even if $\hat{n} >> n$, the awake complexity of our algorithms is not affected at all. The overall number of clock rounds, however, may be affected. In this case n should be replaced by $\hat{n}$ in the clock complexity bounds. In some of our algorithms, however, the dependency on n and $\hat{n}$ can be made as mild as the log-star function. See Section 4.

The main efficiency measures in the Sleeping Model. The measurements for the performance of an algorithm in the sleeping model were first mentioned by Chatterjee, Gmyr and Pandurangan in [9]. For a distributed algorithm with input graph $G = (V, E)$ in the sleeping model, two types of complexity measurements are defined. One is *node-averaged awake complexity* in which, for a node $v \in V(G)$, define $a(v)$ as the number of rounds v spends in the "awake" state until the end of the algorithm. The node-average awake complexity is the average $\frac{1}{n}\sum_{v \in V(G)} a(v)$.
The second efficiency measurement is the *worst-case awake complexity.* This is defined as $\max_{v \in V(G)} a(v)$ and is a stronger requirement than the node-averaged awake complexity. In this paper we focus entirely on the worst-case efficiency measurement.

1.3 Our Techniques

1.3.1 Upper Bound

Our main technical tool is the construction of a *Distributed Layered Tree.* (We denote it shortly by *DLT.*) A DLT is a rooted tree where the vertices are labeled, such that each vertex has a greater label than that of its parent, according to a given order. Moreover, in a DLT each vertex knows its own label and the label of its parent. This knowledge of the label of the parent is not trivial in the sleeping model since passing this information between parent and child requires both of them to be in an awake state. Therefore, this knowledge and hierarchy of IDs throughout the tree makes DLTs are very powerful structures in the sleeping setting. Indeed, once such a tree is computed, the information of the entire graph can be learned by all vertices within $O(1)$ awake complexity, as follows. For a non-root vertex $v \in V(G)$, let $L(v)$ and $L(p(v))$ be the labels of v and the parent of v in the DLT, respectively. Each non-root vertex $v \in V(G)$ is awake only in rounds $L(p(v))$ and $L(v)$. The root r awakes only in round $L(r)$. This way the root is able to perform a broadcast to all the vertices of the tree. Each vertex $v \in V(G)$ receives the information from the root in round $L(p(v))$ (this information has arrived to the parent of v in an earlier stage) and passes it to its children in round $L(v)$. Indeed, in this round v and all its children are awake. In a similar way, a convergecast in the DLT can be performed. We choose some label n' which is greater than all vertex labels in the DLT. Each vertex $v \in V(G)$ awakens in rounds $n' - L(p(v))$ and $n' - L(v)$. This way, information from the leaves propagates towards the root. In round $n' - L(v)$ a vertex v receives information from all its children, and in a later stage, in round $n' - L(p(v))$, the vertex forwards the information to its parent. Note that indeed $n' - L(v) < n' - L(p(v))$, since $L(v) > L(p(v))$, according to the definition of a DLT. A formal proof for the running time of the broadcast and convergecast procedures in a DLT can be found in Lemma 2.1 in Section 2.

Thus, it becomes possible to perform broadcast and convergecast in the tree within 4 awake rounds per vertex. A broadcast and convergecast in a tree allows each vertex to obtain the entire information stored in the tree. Since the tree spans the input graph, the entire information of the graph is obtained. Then, any decidable problem can be solved using the same deterministic algorithm internally in all vertices of the graph. Finally, each vertex deduces its part in the solution. This execution that is performed internally, does not require any additional rounds of distributed communication and is considered as a single awake round in terms of the sleeping model. To summarize this discussion, a DLT makes it possible to solve any decidable problem within 5 awake rounds per vertex.

The ability to solve any decidable problem within a constant awake complexity suggests that the computation of a DLT is an ultimate goal in the sleeping setting. Thus establishing the efficiency of this construction is of great interest. We construct a DLT within $O(\log n)$ awake rounds as follows. We begin with n singleton DLTs, where each vertex of the input graph is a DLT in a trivial way. Then, we perform $O(\log n)$ connection phases in which the trees are merged. Each phase requires at most $O(1)$ awake rounds from each vertex. The number of DLT trees in each phase is at least halved. After $O(\log n)$ phases, a single tree remains. This DLT contains all the vertices of the input graph. Thus, it is the DLT of the entire input graph G.

The high-level idea of our algorithm is somewhat similar to the celebrated algorithm of Gallager, Humblet and Spira for minimum spanning trees [18]. But the construction is fundamentally different. While GHS finds an *existing* subgraph that is an MST, our technique gradually builds trees that contain *new data*. These are new ID assignments that make it possible to have progress with trees formation. In each iteration trees are merged and IDs are reassigned, until a single DLT of the entire input graph is achieved. This tree has the desired IDs, according to the definition of the DLT, as a result of the ID recomputation made in each iteration.

1.3.2 Lower Bound

Once we establish an upper bound on the awake complexity of DLT, we turn to examining lower bounds. We note that an ordinary lower bound technique may not work for the sleeping setting. This is because the standard techniques in the distributed setting deal with what information can be obtained within a certain number of rounds. That is, within r rounds, for an integer r, each vertex can learn its r-hop neighborhood. Then arguments of indistinguishably of views are used. (That is, vertices that must make distinct decisions are unable to do that, if their r-hop-neighborhoods are identical. In this case, r rounds are not sufficient to solve a certain problem.) However, such arguments do not work in the sleeping setting. Indeed, within $O(1)$ awake rounds the entire graph can be learned on certain occasions. Thus, algorithms with r awake rounds are not limited to obtaining knowledge of r-hop-neighborhoods.

As a consequence of the latter phenomenon, we investigate alternative ways to prove lower bounds. We introduce a quite powerful technique that allows one to transfer lower bounds on message complexity into lower bounds for rounds in the sleeping setting. Indeed, if tn messages must be sent in a ring network to solve a certain problem, for an integer $t > 1$, then $t/2$ awake rounds are required for any algorithm that solves the problem in the sleeping setting. Otherwise, if $t' < t/2$ awake rounds are possible, all the messages in each round can be concatenated, and thus each vertex sends up to t' messages to each of its two neighbors in the ring, during the t' rounds it is awake. The number of messages per vertex becomes $2t' < t$, and the overall number of messages passed is thus smaller than tn, which contradicts the assumption that at least tn messages must be sent. A formal proof for this claim can be found in Lemma 3.1 in Section 3.

We employ this idea with the known lower bound of $\Omega(n \log n)$ for message complexity of leader election in rings [17]. Since a DLT allows, in particular, to solve leader election, we deduce that $\Omega(\log n)$ awake rounds are required. Otherwise, it would be possible to solve leader election in rings within fewer than $\Theta(n \log n)$ messages. We note that the lower bound of [17] holds for a given number of rounds assuming the IDs are sufficiently large. Our upper bounds on awake complexity, on the other hand, do not rely on the range of IDs, but only on the number of vertices in the graph. No matter how large the IDs are, on an n-vertex graph the awake complexity for constructing a DLT is $O(\log n)$. The overall number of clock rounds (awake and asleep) do depend on the range of identifiers, but the dependency can be made as low as the log-star function, by using the coloring algorithm of Linial [24]. Our algorithm is applicable also with proper coloring of components, not necessarily with distinct IDs. Consequently, for any given number of clock rounds (awake and asleep), which upper bounds the ordinary running time of an algorithm, there exists a sufficiently large range of IDs, such that the awake complexity $O(\log n)$ of our algorithm is tight.

1.3.3 Improved Upper-Bound for O-LOCAL problems

O-LOCAL problems are those that can be solved sequentially, according to an acyclic orientation provided with the input graph, such that each vertex decision is made after all vertices emanating from it have made their own decisions, and as a function of these decisions. (Note that directing edges from endpoints of smaller IDs to larger IDs provides such an orientation.) For these kind of problems, we employ a technique that is quite different from that of a DLT, and obtain awake complexity of $O(\log \Delta + \log^* n)$. We still employ a tree construction, but this time it is more sophisticated than a DLT. On the other hand, it is constructed internally by each vertex, and is the same in all vertices. The algorithm starts by a distributed computation of an $O(\Delta^2)$-coloring of the input graph within $O(\log^* n)$ time. Then, each vertex constructs internally a binary search tree, whose leaves are the possible $O(\Delta^2)$-colors. (The colors are not consecutive, and inner nodes have integer values between the values of the colors.) Next, each vertex decides to wake-up in the rounds whose numbers appear in the path from the leaf of their color and the root. We prove that one of these rounds occurs after all neighbors of smaller colors have made their decisions. Moreover, by that round these vertices have communicated their decision to the vertex. Consequently, it may compute its own decision. Since the depth of the tree is $O(\log \Delta)$, this requires only $O(\log \Delta)$ awake-rounds per vertex.

1.4 Related Work

The distributed $\mathcal{LOCAL}$ model was formalized by Linial in his seminal paper [24] from 1987. This paper also provides a deterministic $O(\Delta^2)$-coloring algorithm with $O(\log^* n)$ round-complexity, as well as a matching lower bound. Since then, a plethora of distributed graph algorithms has been obtained in numerous works. See, e.g., the survey of [27] and references therein.

The sleeping setting has been intensively studied in the area of Computer Networks [10, 25, 28, 29]. In Distributed Computing the problem of broadcast in sleeping radio networks was studied by King, Phillips, Saya and Young [22]. The problem of clock synchronization in networks with sleeping processors and variable initial wake-up times was studied by Bradonjic, Kohler and Ostrovsky [7], and by Barenboim, Dolev and Ostrovsky [2]. A special type of the sleeping model, in which processors are initially awake, and eventually enter a permanent sleeping state, was formalized by Feuilloley [13, 14]. An important efficiency measurement in this setting is the *vertex-averaged* awake complexity. This setting was further studied by Barenboim and Tzur [6], who obtained various symmetry-breaking algorithms with improved vertex-averaged complexity.

The awake complexity of various problems has been also studied in radio models of general graphs (rather than unit disk graphs). In particular, several important results were achieved by Chang et al. in PODC'18 [8]. That work considered Broadcast and related problems in several radio models, that can be seen as the sleeping model with additional restrictions. Specifically, in the model that is the closest to the sleeping model, the vertices are able to either transmit or listen in an awake round, but not both. (In other words, this is a half-duplex communication model, while the sleeping model is full-duplex.) There are also even more restricted models studied in [8], in which vertices cannot receive messages from multiple neighbors in parallel.

Since the results of Broadcast [8] are applicable to the sleeping model, and they are the state-of-the-art even in this model that we consider in the current paper, a comparison between them and our results is in place. The Broadcast algorithm of [8] with the best deterministic awake complexity has efficiency $O(\log n \cdot \log N)$, where $N \geq n$ is the largest identifier. Our results, on the other hand, provide a deterministic Broadcast algorithm in the sleeping setting with awake complexity of $O(\log n)$. This is at least a quadratic improvement in the sleeping setting. We stress that our awake complexity is not affected by the size of identifiers, and remains $O(\log n)$, no matter how large N is. The Broadcast algorithm of [8] constructs trees that partition the graph into layers, but these trees are very different from our DLTs, both in their structure and in the techniques for achieving them. Specifically, in [8] each vertex in layer $i = 1, 2, ...$ in the tree has a neighbor of layer $i - 1$. On the other hand, a DLT does not necessarily have this property (This is because layer i in a DLT consists of all vertices labeled by i in the tree, which are not necessarily at distance i from the root.) In addition, the tree construction in [8] is based on ruling sets, while our techniques are considerably different.

The class **O-LOCAL** of problems that we mentioned in Section 1.3.3 is inspired by the class **P-SLOCAL** which was first defined by Ghaffari, Kuhn and Maus [20]. This class consists of all problems that can be solved as follows. Given an acyclic orientation, the output of each vertex is determined sequentially, according to the orientation, after vertices on outgoing edges have made their decisions. The decision of each vertex is based on information from a polylogarithmic radius around it. The **O-LOCAL** class is similar to the **P-SLOCAL** class, except that instead of examining a polylogarithmic-radius neighborhood around a vertex, its neighbors on outgoing edges are examined. (And, more generally, all vertices on consistently oriented paths emanating from a vertex are inspected.)

2 Distributed Layered Trees

In this section we describe our method with which one can solve any decidable distributed problem in the sleeping model. We describe the construction of a certain kind of a spanning tree, called a *Distributed Layered Tree*, defined as follows. Each vertex v in the tree is labeled with a label $L(v)$. These labels must have some predefined order, such that they can be mapped to natural numbers. The labeling is such that the label of each vertex, besides the root, is larger than the label of its parent, and each vertex knows the label of its parent. These two requirements of the spanning tree allow us to perform broadcast across the spanning tree in a fashion where each vertex is awake for exactly 2 rounds. This is also true for a convergecast procedure. Consequently, given such a tree, the root can learn the entire input graph, compute a solution for any decidable problem internally, and broadcast it to all vertices, all within a constant number of awake rounds. This is done in the following way. For a broadcast procedure, we start with a message from the root of the tree through the tree, where each vertex v is awake for just 2 rounds. Namely, v awakes in round $L(v)$ and round $L(p(v))$, where $p(v)$ is the parent of v in the spanning tree. In other rounds v is asleep. This ensures that a message sent from the root will propagate in the tree and eventually arrive to all vertices of the graph while having each vertex awaken exactly twice. In the same manner we perform the convergecast procedure, where each child v sends its message to its parent at the round $n' - L(p(v))$ for some $n' \geq n$. Again we have each vertex v active for only two rounds, specifically, $n' - L(v)$ and $n' - L(p(v))$.

Using this method one can have the root of the spanning tree compute the solution for any decidable problem $\mathcal{P}$ deterministically and broadcast this solution back through the tree to all the vertices in the input graph G with $O(1)$ worst-case complexity in the sleeping model. Therefore, our main goal is obtaining a distributed sleeping algorithm for computing such a layered tree. We begin with a formal definition of notations and the definition of a Distributed Layered Tree. Note that we refer here to lexicographic order. For sake of definition we do not limit or define this lexicographic order since it does not matter for the definition of a DLT. One can build a DLT with any order that can be mapped to natural numbers. We define a lexicographic order that serves our purposes in Section 2.1.

Vertex Label $L(v)$. The label of a vertex v is denoted by $L(v)$. The labels are taken from a range of labels which has a lexicographic order.

Tree Label $L(\mathcal{T})$. The label of a tree $\mathcal{T}$ is denoted by $L(\mathcal{T})$. The labels are taken from a range of labels which has a lexicographic order. The label of a tree is defined to be the label of the root of the tree. Hence, that label can always be found in the memory of the root.

▶ **Definition 2.1** (A Distributed Layered Tree (DLT).). A DLT is a rooted oriented labeled spanning tree with two properties, with respect to some lexicographical order:

1. For each vertex v and a parent $p(v)$, the label of v is greater than the label of $p(v)$ in the lexicographical order.
2. v has knowledge of the label of $p(v)$.

As we show in the next lemma, DLTs are useful for distributing data across the graph in an efficient way.

▶ **Lemma 2.1.** *Given a DLT, the procedures of broadcast and convergecast across the entire tree take exactly 2 rounds each in the sleeping model.*

Proof.

Broadcast. Each vertex v is awake in rounds $L(v)$ and $L(p(v))$. As a vertex v awakes in round $L(v)$ and broadcasts a message, each of its children u in the tree are awake in round $L(p(u)) = L(v)$ and thus can receive the message from its parent. Therefore, a message sent by the root propagates throughout the tree until it reaches all leaves.

Convergecast. Let n' be an integer, such that $n' > L(v)$, for all $v \in V$, and n' is known to all vertices. Each vertex v is awake in rounds $n' - L(v)$ and $n' - L(p(v))$. If a child v has a message to pass to its parent $p(v)$, it awaits round $n' - L(p(v))$ and then v sends the message to $p(v)$. In that round $p(v)$ is awake and ready to receive the messages from all its children. Each vertex v already has knowledge of the subtree rooted at it, since for each vertex u in the subtree in which v is the root, $L(u) > L(v) > L(p(v))$ and thus the round $n' - L(u)$ comes before the round $n' - L(p(v))$. Thus the message propagates up the tree all the way to the root. ◀

We note that in the proof above we require for an integer n' to be larger than all labels of all vertices in the tree. Since the labels are required to be taken from a range with lexicographic order, and we have knowledge of the size of this range, n' can be chosen appropriately. We describe the lexicographic order in Section 2.1, as well as describing how each vertex has knowledge of ranges from which labels are selected.

2.1 The Connection Phases

Our algorithm for constructing a DLT starts with a graph where each vertex is considered to be a singleton tree. The initial label of each such tree $\mathcal{T}_v$, $v \in V$, is determined by the ID of its vertex v as follows. $L(\mathcal{T}_v) = ID(v)$. The vertex label is set as $L(v) = \langle ID(v), 0\rangle$. (Two coordinates are used, since trees are going to be merged and have many vertices. Then the left-hand coordinate is going to be the same for all vertices in a tree, while the right-hand coordinates may differ. Also, distinct trees will have distinct left-hand coordinate.) Each of these n singleton trees is a DLT in a trivial way. Our goal is merging these trees in stages, so that eventually a single DLT remains that spans the entire input graph. Assume we have a forest of DLTs. Initially, we have a forest of n single-vertex trees. During the connection phases, we enforce a rule regarding the representation of the labels of the vertices. Let $\mathcal{T}$ be a tree in our forest. The DLT label of $\mathcal{T}$, $L(\mathcal{T})$ is an integer number. The label of each vertex $v \in \mathcal{T}$ is set as $\langle L(\mathcal{T}), l\rangle$, where l is a number assigned to v, as described in Section 2.1.1. In what follows we describe the ordering of the vertex labels. The left coordinate is considered to be the more significant one among the two. That is, $\langle a, b\rangle < \langle c, d\rangle$ iff $a < c$ or ($a = c$ and $b < d$). Note that the requirements on labels hold. That is, the labels have an ordering and the root of the tree can deduce the tree label from its own label by referring to the first coordinate. Next, we describe how our algorithm produces connections between the trees. We do this in two stages.

2.1.1 Connection Stage One – Several DLTs into a single DLT

In this stage our goal, for each tree $\mathcal{T}$, is finding an edge (u, w) that connects $\mathcal{T}$ to a neighbor DLT $\hat{\mathcal{T}}$, such that $L(\mathcal{T}) > L(\hat{\mathcal{T}})$. In this sense, $u \in \mathcal{T}$ and $w \in \hat{\mathcal{T}}$. Using this edge we connect $\mathcal{T}$ with $\hat{\mathcal{T}}$, such that w becomes the parent of u. In the case that $\mathcal{T}$ is a single vertex v, we simply choose the neighbor of v in G with a label smaller than that of v. We note that there may be a case that no such edge is found, since $\mathcal{T}$ is a DLT with local minimum label. We handle such a case in Section 2.1.2. If $\mathcal{T}$ contains more than one vertex, we perform a convergecast from all the neighbors of all vertices in $\mathcal{T}$ to the root of $\mathcal{T}$. Consequently, the root of $\mathcal{T}$ learns the structure of $\mathcal{T}$ and the set of edges that connect $\mathcal{T}$ with other trees.

The root chooses the edge (u, w) which connects $\mathcal{T}$ to a vertex $w \in \hat{\mathcal{T}}$, where $\hat{\mathcal{T}} \neq \mathcal{T}$ is a neighboring DLT of $\mathcal{T}$. As explained above, the choice is made such that $L(\hat{\mathcal{T}}) < L(\mathcal{T})$. Recall that at this point $L(\hat{\mathcal{T}})$ appears in the first coordinate of all the labels of the vertices in $\hat{\mathcal{T}}$, and hence the root of $\mathcal{T}$ has knowledge of all the labels of its neighboring DLTs.

Internally, the root of $\mathcal{T}$ calculates a new label arrangement of the vertices of $\mathcal{T}$, such that the vertex u that was chosen above (that is connected to $w \in \hat{\mathcal{T}}$) becomes the new root, and $\mathcal{T}$ remains a DLT, under the new label arrangement. This requires a new orientation of $\mathcal{T}$. This new oriented tree is denoted $\mathcal{T}'$. The label arrangement of $\mathcal{T}'$ is obtained in the following way. Note that the DLT label of $\mathcal{T}$ is $L(\mathcal{T})$. The new root u sets its label to be $\langle L(\mathcal{T}), 0\rangle$. The rest of the vertices in $\mathcal{T}'$ are assigned labels in the following way. Each vertex v is assigned the label $\langle L(\mathcal{T}), l\rangle$ where l is the distance of v from u in $\mathcal{T}'$. Specifically, l is the level of v in $\mathcal{T}'$. It follows that each level of $\mathcal{T}'$ has labels smaller than the labels in the next level of the tree. Once this internal computation is done, $r_{\mathcal{T}}$ sends the resulting label arrangement in a broadcast procedure to all vertices in $\mathcal{T}'$. Note that u becomes the new root, instead of $r_{\mathcal{T}}$, from now until further notice.

But now we are posed with the problem that neighbors of the vertices of the tree $\mathcal{T}'$ do not know the labels of their neighbors, which is required for the next connection phase as well as the second requirement for a DLT. This is solved by awakening the entire graph for one round. Each vertex sends its knowledge to all of its neighbors while receiving the knowledge from all of its neighbors in the same round. Then all vertices of the graph switch to asleep status. The round in which this happens is determined by all vertices before the beginning of the execution. Once all vertices determine the starting time of such a phase, they all wake up after cn' rounds, for a constant c, such that cn' bounds from above the execution time of this phase. In the following lemma we prove that this connection procedure connects several DLTs into one DLT. The proof is available in the full version of this paper [5].

▶ **Lemma 2.2.** *Let C be a connected component in our graph produced at this stage. Then C is a DLT.*

We would like to note here that at the end of each connection phase each DLT C'' in the new forest has a distinct DLT label. This is due to the fact that the DLT label of C'' is set from one of the labels of the trees composing C''. If each DLT had a distinct label at the start of the connection phase, it is clear we preserved this property also once the connection phase is done. Thus, it is also true in the start of the next connection phase. Note that we start the algorithm with each DLT having a distinct DLT label since we set those labels according to the IDs of the vertices of G.

2.1.2 Connection Stage Two – Connecting Local Minimums

We now make a second connection step as part of the connection phase. The motivation in this step is that there might be trees where the DLT labels are local minima and thus those trees did not connect to any other tree. They might also not have been chosen by any other tree. If such is the case, these trees made it through stage one of the connection phase without connecting to any other component.

We would like to at least halve the number of trees in each phase and have at most $O(\log n)$ connection phases, we connect these local-minimum label trees to other components. Let T_m be such a local-minimum label tree. First, we would like to verify that T_m indeed has no connections to other components from the previous stage. If some other component has selected T_m earlier, we no longer need to handle it, even though T_m is a minimum label tree. We only need to handle trees with no connection to other trees. We check this by performing a convergecast in each connected component sending signals from the leaves to the root. If r_{T_m} received a signal from vertices that previously did not belong to T_m then T_m is not a problematic tree and was chosen by another tree in the first stage. This signal can be, for example, the label of a leaf. Since the label contains the root of the tree to which the leaf belongs (before the trees are merged), then r_{T_m} can deduce that the leaf was not part of T_m before the first stage of the connection phase. In this case we do nothing with T_m.

The other option is where T_m was not connected to any other component. In this case, we add such a connection. To this end, r_{T_m} chooses an edge (v, u) to connect to a DLT C arbitrarily. Since it is the only edge connecting it to other DLTs, the result is a tree. A simple move now would be to make C a subtree of the DLT T_m. But we note that T_m might not be the only tree that arbitrarily chose to connect to C. Since there can be more than one local minimum DLT that chooses C, we need to make C the parent of all those DLTs which chose to connect to it. Otherwise, C will become the child-DLT of more than one DLT which breaks the structure of a directed tree we aspire for.
Therefore, in order for the result to be a DLT we need to make T_m the sub-DLT of C instead. That is, given that (v, u) is the edge connecting T_m to C, we turn v into the root of T_m and make v the child of u in $C \cup T_m$. Doing so poses a problem since $L(v) < L(u)$ and this violates the requirements for a DLT in the resulting component. We solve this by waking up the entire graph for a single round and have v and u exchange information.

After this round, the information about the local-minimum label trees that asked to join C is located in vertices of the component C. This information is then delivered to the root of C, r_C by a convergecast procedure. r_C performs label reassignment in the same way as in Section 2.1.1. Specifically, a single BFS is computed for all vertices in C and the local minimum label trees T_m that connected to C. Then reassignment of these labels is made according to the levels of the BFS tree. This results in labels of the from $\langle ID(r_C), l\rangle$, such that the first coordinate is the same for all vertices in the connected component and l is the level of the vertex in C. Note that the structure of the labels and the label arrangement is the same as described in Section 2.1.1. Thus, the proof of Lemma 2.2 applies here for the new connected component and its labeling. Therefore, C is a DLT. This completes the description of connection stages. See Appendix C for a pseudocode with a summary of their steps. In what follows, we analyze the algorithm.

2.2 Analysis

We turn to analyse our algorithm for spanning a DLT on the input graph G. We prove several lemmas and conclude with the main result for our scheme. The proof is available in the full version of this paper [5].

▶ **Lemma 2.3.** *Let C be a connected component at the end of a connection phase. Then C is a DLT.*

Next, we analyze the awake complexity of the algorithm. Our claim is that a connection phase halves the number of DLTs in the forest. This is quite straightforward. If a DLT is not connected in stage one of the connection phase, phase two considers it as problematic and makes sure it connects to another DLT. Thus, every DLT connects to another DLT and thus the number of DLTs is at least halved.
The next lemma analyses the performance of a connection phase. The proof is available in the full version of this paper [5].

▶ **Lemma 2.4.** *Each vertex v is awake for at most $O(1)$ rounds in each connection phase.*

We can now conclude the analysis of our DLT spanning algorithm. Since the number of DLTs is at least halved in each phase, there are $O(\log n)$ such phases, and the following theorem is directly obtained from Lemma 2.4.

▶ **Theorem 2.5.** *A DLT for any input graph G can be deterministically computed in $O(\log n)$ awake rounds in the sleeping model.*

As shown in Lemma 2.1, the action of convergecast and the action of broadcast on the resulting spanning tree each require only $O(1)$ time in the sleeping model and thus we can collect the topology of the entire input graph to the root of our spanning tree, calculate the solution to the problem $\mathcal{P}$ deterministically and broadcast the solution to all vertices through our spanning tree, again in $O(1)$ time. This places an upper bound on the class of all decidable problems as we conclude in the following theorem.

▶ **Theorem 2.6.** *Any decidable problem can be solved within $O(\log n)$ worst-case deterministic awake-complexity in the sleeping model.*

3 A Tight Bound for DLT

In this section we prove that the complexity of DLT in the sleeping model is $\Omega(\log n)$. The proof is by a reduction from leader election on rings. For the latter problem, it is known that a certain number of messages must be sent in the network in order to solve it [17]. In what follows we prove that this lower bound on messages implies a lower bound on awake rounds in the sleeping model. Before presenting the proof, we need the following lemma, which demonstrates a connection between the number of messages an algorithm produces and the complexity in the sleeping model.

▶ **Lemma 3.1.** *Any algorithm $\mathcal{A}$ which requires at least $\Omega(\Delta n \log n)$ messages for its execution has an awake complexity of at least $\Omega(\log n)$ in the sleeping model.*

Proof. Let the number of messages that must be sent during the execution of $\mathcal{A}$ be $c\Delta \cdot n \log n$ where $c > 0$ is a constant. We show that there is at least one vertex v that must be awake for at least $c \log n$ rounds. Assume for contradiction that all vertices in G are awake for less than $c \log n$ rounds. Each vertex sends at most Δ messages (one across each adjacent edge) in a single awake round. If more than one message per edge per round is required, all these messages can be concatenated into a single message. Thus, each vertex sends less than $\Delta \cdot c \log n$ messages, and the overall number of messages in the execution is less than $n(\Delta \cdot c \log n) = c\Delta \cdot n \log n$. This is a contradiction. Therefore, there must be a vertex that is awake for $\Omega(\log n)$ rounds. We conclude that the awake round complexity of $\mathcal{A}$ in the sleeping model is also $\Omega(\log n)$. Given that there are at least $\Omega(\Delta n \log n)$ messages and n vertices and at most Δn edges, on average, each vertex is awake for at least $c \log n$ rounds. Thus, the running time of $\mathcal{A}$ (in the worst case and average case) is at least $\Omega(\log n)$. ◀

▶ Remark. An algorithm $\mathcal{A}$ that requires $\Omega(\Delta n \log n)$ messages has an awake complexity of $\Omega(\log n)$, not only in the worst vertex, but also on average over the vertices. (Such an average complexity is referred to as *vertex-averaged complexity* [9].) Indeed , if the vertex-averaged awake complexity is $o(\log n)$ then the sum of awake rounds for all vertices is $o(n \log n)$, and the number of messages is $o(\Delta n \log n)$, according to the proof of Lemma 3.1.

Next, we employ Lemma 3.1 in order to prove that DLT requires $\Omega(\log n)$ complexity in the sleeping model. We show this for a ring graph by a reduction from the leader election problem.

▶ **Theorem 3.2.** *Let $t > 0$ be an arbitrarily large integer, and $\mathcal{A}$ any deterministic algorithm for the DLT problem, which requires t rounds in the $\mathcal{LOCAL}$ model. Then there is an ID assignment from a sufficiently large range, as a function of t, such that $\mathcal{A}$ requires $\Omega(\log n)$ awake-complexity in the sleeping model.*

Proof. The proof is by contradiction. Assume that there is an algorithm $\mathcal{A}$ with awake-round complexity of $o(\log n)$, overall complexity $t > 0$, for ID assignment from an arbitrarily large range. Then $\mathcal{A}$ uses at most $o(n\Delta \log n)$ messages (see Lemma 3.1). Let C be an n-vertex cycle graph. The maximum degree of C is $\Delta = 2$. We execute $\mathcal{A}$ on C in the ordinary (not-sleeping) $\mathcal{LOCAL}$ model. We obtain a DLT of C within t rounds. Now, the root can be elected as the leader, and the other vertices know that they are not the root. In a DLT they also know the ID of the root. Thus, we have an algorithm for leader election in the $\mathcal{LOCAL}$ model which employs at most $o(n \log n)$ messages.

According to [17], the leader election problem requires $\Omega(n \log n)$ messages, if vertex IDs are chosen from a set of sufficiently large size $R(n, t)$, where R is the Ramsey function and t is the running time of the algorithm. This is a contradiction. ◀

It follows that any problem whose solution can be used to elect a leader within $o(\log n)$ additional awake rounds requires $\Omega(\log n)$ awake-complexity. We denote the class of such problems by **DLT-hard** problems. Theorems 2.5 and 3.2 directly give rise to the following corollary.

▶ **Theorem 3.3.** *The class of DLT-hard problems has a deterministic complexity tight bound of $\Theta(\log n)$ in the sleeping model.*

An alternative proof. The following alternative proof was suggested by an anonymous referee. A lower bound of [8] shows that broadcast on a path requires $\Omega(\log n)$ awake rounds. Their model allows each vertex either to transmit or listen in a round. However, the proof goes through even when vertices may transmit and listen in the same time. Consequently, a DLT requires $\Omega(\log n)$ awake rounds as well. The lower bound proof of [8] applies also to randomized algorithms.

▶ Remark. Note that in Section 2.1 we showed that the DLT problem is complete in the class of decidable problems and Theorem 3.3 states it is $\Omega(\log n)$-hard under this class.

4 Solving Oriented-Local Problems

In this section we devise an algorithm for solving a class of *Oriented-Local* problems. This class contains all problems which, given an acyclic orientation on the edge set of the graph, can be solved as follows. Each vertex awaits all neighbors on outgoing edges to produce an output, and then computes its own output as a function of the outputs of these neighbors. (Vertices with no outgoing edges produce an output immediately.) We define this class formally.

▶ **Definition 4.1.** The class of **1-hop Oriented Local Problems (1-O-LOCAL)** consists of all problems that, given an acyclic orientation μ on the edge set of G, can be solved in the following way. Let v be a vertex in G. Let U be the set of neighbors of v in its 1-hop neighborhood which precedes v in the orientation μ, i.e., the vertices connected by outgoing edges from v. Let $s(p)$, for $p \in U$, be the solution of the problem. Then, v can internally calculate $s(v)$ with the knowledge of $\{s(p) \mid p \in U\}$.

The class of **Oriented Local Problems (O-LOCAL)** is a generalization of 1-O-LOCAL, where the set U contains all vertices on paths that emanate from v, rather than v's immediate neighbors on such paths.

As one can tell, a solution for a problem in this class depends on a given orientation. Such orientation can be calculated or given as an input to the algorithm. In this work we assume that no orientation is given and we are forced to calculate one as part of the solution. We note that MIS and $(\Delta + 1)$-vertex-coloring are examples of well-studied problems which fall in the class of **1-O-LOCAL** problems.

We start with an algorithm for $O(\Delta^2)$-vertex-coloring in $O(\log^* n)$ time [24]. This gives us an orientation of the edges where we orient edges in descending order, i.e., each edge is oriented towards the endpoint of a smaller color. We have all vertices in G awake during the entire coloring algorithm. Let $q = O(\Delta^2)$ be an upper bound on the number of colors of the algorithm of Linial such that q is a power of 2. At the next stage each vertex v builds a binary search tree internally. The size of the tree is $2q - 1$. The root of the tree receives the label in the middle of the range $[2q - 1]$, which is q. Now we have $q - 1$ values in each side of the tree. Specifically, $\{1, \ldots, q - 1\}$ for the left subtree and $\{q + 1, \ldots, 2q - 1\}$ for the right subtree. We choose the middle of the range $\{1, \ldots, q - 1\}$ for the left child of the root and the middle of the range $\{q + 1, \ldots, 2q - 1\}$ for the right child of the root. We continue this recursively, so that each node of the tree obtains a unique value from $[2q - 1]$.

Now we recolor the vertices of the input graph using the following mapping. The recoloring is performed by all vertices in parallel, with no communication whatsoever. We map the elements from $[q]$ to the set of values appearing in the leaves of the binary tree. The mapping is the same in all vertices. Specifically, for each $i \in [q]$, the i-th element is mapped to the label of the i-th left-most leaf of the tree. Consequently, all vertices that were initially colored by the color i switch their color to the label of the $i$the leaf in the tree. Note that each pair of neighbors select distinct leaves of the tree, since their original colors are distinct. Therefore, the coloring after the mapping is proper as well.

Next, we switch to the sleeping state for all the vertices in the graph, and start solving a given **1-O-LOCAL** problem P. For the sake of simplicity, we proceed with the problem of MIS, but our method can be applied to any **1-O-LOCAL** problem, as would be obvious from the description of the algorithm. The scheme is as follows. Each vertex v employs its color in the $O(\Delta^2)$-coloring, and a respective leaf in the binary tree, whose value equals the color of v. Let R be the path from the leaf of the color of v to the root of the binary tree. Let $r(v) = \{r_1, \ldots, r_k\}$ be the values appearing in R. We denote $r' = r_1$. Note that some values in $r(v)$ may be greater than r', while other values may be smaller than r'. Then v awakes at each round $r_i \in r(v)$, and sends a message to its awake neighbors about its state, e.g., whether it is in the MIS, not in the MIS or undecided. It also receives such messages from its awake neighbors in these rounds. Recall that $r' = r_1$ is the round number that is equal to the color of v. In round r' the vertex v makes a decision if to join the MIS or not according to the information received from outgoing edges. The neighbors on such edges have smaller colors, and thus have made a decision before round r'. See Appendix A for an example of a colored binary tree. The following lemma proves that v has all the information from vertices of lower colors when round r' arrives. See the proof in Appendix B.

▶ **Lemma 4.1.** *At round r', which is mapped to the color of v, all vertices with colors smaller than that of v have already made a decision. Furthermore, their decisions have been passed to v in a previous round.*

For a problem P in **1-O-LOCAL**, a vertex v can make its decision in round r'. For example, the following decisions are made in some well-studied **1-O-LOCAL** problems: For MIS, v joins the MIS if all neighbors with lower colors are not in the MIS. For $(\Delta+1)$-vertex-coloring, v chooses a new color from the palette $[\Delta+1]$ which is not yet chosen as a new color by its neighbors with lower old-colors (i.e., colors according to the initial orientation).

The depth of a binary tree with $O(\Delta^2)$ leaves is at most $O(\log\Delta)$. Thus, the size of a path R from a leaf to the root is at most $O(\log\Delta)$. The vertex v only awakens in rounds corresponding to keys appearing along R, and thus v awakens in at most $O(\log\Delta)$ rounds. This provides us with the complexity of our algorithm in the following theorem.

▶ **Theorem 4.2.** *Any* **1-O-LOCAL** *problem can be solved in* $O(\log\Delta+\log^* n)$ *deterministic awake-complexity in the sleeping model.*

Note that each vertex is able to accumulate all information received from outgoing neighbors and pass it later to incoming neighbors, when these neighbors ask it for its output. Consequently, each vertex learns the information from all vertices that emanate from it in the orientation. Thus, each vertex is able to produce an output not only as a function of its outgoing-neighbors outputs, but as a function of all output of vertices that emanate from it. In other words, any **O-LOCAL** problem can be solved this way. Hence, we obtain the following corollary.

▶ **Corollary 4.3.** *Any* **O-LOCAL** *problem can be solved in* $O(\log\Delta+\log^* n)$ *deterministic awake-complexity in the sleeping model.*

In addition to the above results we also obtained further results. They are omitted from this version of the paper, due to space limitations. We refer the reader to the full version of this paper for a detailed description of these results [5].

5 Conclusion

In this work we investigated the strength of Distributed Layered Trees in the sleeping model. We showed that the computation of such trees is complete and thus any decidable problem can be solved within $O(\log n)$ awake complexity. This raises the question of finding non-trivial sub-classes of decidable problems which one can solve in a more efficient way than using a DLT. We address this question by defining the **O-LOCAL** class of problems and showing that it indeed can be solved more efficiently in the sleeping model. Since the $\mathcal{CONGEST}$ model is of great interest in the field of distributed networks, we investigated it as well, and obtained a class of problems that can be solved within logarithmic awake complexity by using only messages of logarithmic size. Another important aspect is the number of ordinary clock rounds of an algorithm with good awake complexity. While our simpler version of the algorithm has quadratic complexity of clock rounds, the more sophisticated variant gets closer to the optimal $\Theta(n)$ rounds. Overall, we showed the strength of the sleeping model and the possibility of a significant energy conservation for distributed networks.

References

1 Alkida Balliu, Fabian Kuhn, and Dennis Olivetti. Distributed edge coloring in time quasi-polylogarithmic in delta. In *PODC '20: ACM Symposium on Principles of Distributed Computing, Virtual Event, Italy, August 3-7, 2020*, pages 289–298. ACM, 2020. doi:10.1145/3382734.3405710.

2 Leonid Barenboim, Shlomi Dolev, and Rafail Ostrovsky. Deterministic and energy-optimal wireless synchronization. *ACM Trans. Sens. Networks*, 11(1):13:1–13:25, 2014. doi:10.1145/2629493.

3 Leonid Barenboim and Michael Elkin. Deterministic distributed vertex coloring in polylogarithmic time. *J. ACM*, 58(5):23:1–23:25, 2011. doi:10.1145/2027216.2027221.

4 Leonid Barenboim, Michael Elkin, and Tzalik Maimon. Deterministic distributed (delta + o(delta))-edge-coloring, and vertex-coloring of graphs with bounded diversity. In *Proceedings of the ACM Symposium on Principles of Distributed Computing, PODC 2017, Washington, DC, USA, July 25-27, 2017*, pages 175–184. ACM, 2017. doi:10.1145/3087801.3087812.

5 Leonid Barenboim and Tzalik Maimon. Deterministic logarithmic completeness in the distributed sleeping model. *CoRR*, abs/2108.01963, 2021. arXiv:2108.01963.

6 Leonid Barenboim and Yaniv Tzur. Distributed symmetry-breaking with improved vertex-averaged complexity. In *Proceedings of the 20th International Conference on Distributed Computing and Networking, ICDCN 2019, Bangalore, India, January 04-07, 2019*, pages 31–40. ACM, 2019. doi:10.1145/3288599.3288601.

7 Milan Bradonjic, Eddie Kohler, and Rafail Ostrovsky. Near-optimal radio use for wireless network synchronization. *Theor. Comput. Sci.*, 453:14–28, 2012. doi:10.1016/j.tcs.2011.09.026.

8 Yi-Jun Chang, Varsha Dani, Thomas P. Hayes, Qizheng He, Wenzheng Li, and Seth Pettie. The energy complexity of broadcast. In Calvin Newport and Idit Keidar, editors, *Proceedings of the 2018 ACM Symposium on Principles of Distributed Computing, PODC 2018, Egham, United Kingdom, July 23-27, 2018*, pages 95–104. ACM, 2018. doi:10.1145/3212734.3212774.

9 Soumyottam Chatterjee, Robert Gmyr, and Gopal Pandurangan. Sleeping is efficient: MIS in $O(1)$-rounds node-averaged awake complexity. In *PODC '20: ACM Symposium on Principles of Distributed Computing, Virtual Event, Italy, August 3-7, 2020*, pages 99–108. ACM, 2020. doi:10.1145/3382734.3405718.

10 Jing Deng, Yunghsiang S. Han, Wendi Rabiner Heinzelman, and Pramod K. Varshney. Scheduling sleeping nodes in high density cluster-based sensor networks. *Mob. Networks Appl.*, 10(6):825–835, 2005. doi:10.1007/s11036-005-4441-9.

11 Cynthia Dwork, Joseph Y. Halpern, and Orli Waarts. Performing work efficiently in the presence of faults. *SIAM J. Comput.*, 27(5):1457–1491, 1998. doi:10.1137/S0097539793255527.

12 Laura Marie Feeney and Martin Nilsson. Investigating the energy consumption of a wireless network interface in an ad hoc networking environment. In *Proceedings IEEE INFOCOM 2001, The Conference on Computer Communications, Twentieth Annual Joint Conference of the IEEE Computer and Communications Societies, Twenty years into the communications odyssey, Anchorage, Alaska, USA, April 22-26, 2001*, pages 1548–1557. IEEE Comptuer Society, 2001. doi:10.1109/INFCOM.2001.916651.

13 Laurent Feuilloley. How long it takes for an ordinary node with an ordinary ID to output? In *Structural Information and Communication Complexity - 24th International Colloquium, SIROCCO 2017, Porquerolles, France, June 19-22, 2017, Revised Selected Papers*, volume 10641 of *Lecture Notes in Computer Science*, pages 263–282. Springer, 2017. doi:10.1007/978-3-319-72050-0_16.

14 Laurent Feuilloley. How long it takes for an ordinary node with an ordinary id to output? *Theor. Comput. Sci.*, 811:42–55, 2020. doi:10.1016/j.tcs.2019.01.023.

15 Manuela Fischer. Improved deterministic distributed matching via rounding. *Distributed Comput.*, 33(3-4):279–291, 2020. doi:10.1007/s00446-018-0344-4.

16 Manuela Fischer, Mohsen Ghaffari, and Fabian Kuhn. Deterministic distributed edge-coloring via hypergraph maximal matching. In *58th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2017, Berkeley, CA, USA, October 15-17, 2017*, pages 180–191. IEEE Computer Society, 2017. doi:10.1109/FOCS.2017.25.

17 Greg N. Frederickson and Nancy A. Lynch. Electing a leader in a synchronous ring. *J. ACM*, 34(1):98–115, 1987. doi:10.1145/7531.7919.

18 Robert G. Gallager, Pierre A. Humblet, and Philip M. Spira. A distributed algorithm for minimum-weight spanning trees. *ACM Trans. Program. Lang. Syst.*, 5(1):66–77, 1983. `doi:10.1145/357195.357200`.

19 Leszek Gasieniec, Erez Kantor, Dariusz R. Kowalski, David Peleg, and Chang Su. Time efficient k-shot broadcasting in known topology radio networks. *Distributed Comput.*, 21(2):117–127, 2008. `doi:10.1007/s00446-008-0058-0`.

20 Mohsen Ghaffari, Fabian Kuhn, and Yannic Maus. On the complexity of local distributed graph problems. In *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2017, Montreal, QC, Canada, June 19-23, 2017*, pages 784–797. ACM, 2017. `doi:10.1145/3055399.3055471`.

21 Michal Hanckowiak, Michal Karonski, and Alessandro Panconesi. On the distributed complexity of computing maximal matchings. In *Proceedings of the Ninth Annual ACM-SIAM Symposium on Discrete Algorithms, 25-27 January 1998, San Francisco, California, USA*, pages 219–225. ACM/SIAM, 1998. URL: `http://dl.acm.org/citation.cfm?id=314613.314705`.

22 Valerie King, Cynthia A. Phillips, Jared Saia, and Maxwell Young. Sleeping on the job: Energy-efficient and robust broadcast for radio networks. *Algorithmica*, 61(3):518–554, 2011. `doi:10.1007/s00453-010-9422-0`.

23 Fabian Kuhn. Faster deterministic distributed coloring through recursive list coloring. In Shuchi Chawla, editor, *Proceedings of the 2020 ACM-SIAM Symposium on Discrete Algorithms, SODA 2020, Salt Lake City, UT, USA, January 5-8, 2020*, pages 1244–1259. SIAM, 2020. `doi:10.1137/1.9781611975994.76`.

24 Nathan Linial. Distributive graph algorithms-global solutions from local data. In *28th Annual Symposium on Foundations of Computer Science, Los Angeles, California, USA, 27-29 October 1987*, pages 331–335. IEEE Computer Society, 1987. `doi:10.1109/SFCS.1987.20`.

25 Miao Peng, Yang Xiao, and Pu Patrick Wang. Error analysis and kernel density approach of scheduling sleeping nodes in cluster-based wireless sensor networks. *IEEE Trans. Veh. Technol.*, 58(9):5105–5114, 2009. `doi:10.1109/TVT.2009.2027908`.

26 Václav Rozhon and Mohsen Ghaffari. Polylogarithmic-time deterministic network decomposition and distributed derandomization. In *Proccedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing, STOC 2020, Chicago, IL, USA, June 22-26, 2020*, pages 350–363. ACM, 2020. `doi:10.1145/3357713.3384298`.

27 Jukka Suomela. Survey of local algorithms. *ACM Comput. Surv.*, 45(2):24:1–24:40, 2013. `doi:10.1145/2431211.2431223`.

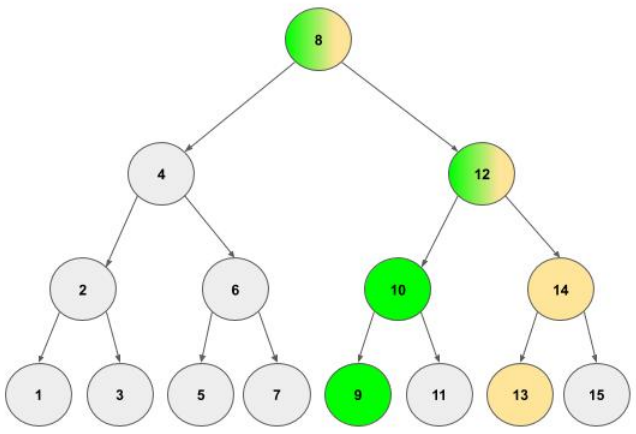
28 Chafiq Titouna, Abdelhak Mourad Guéroui, Makhlouf Aliouat, Ado Adamou Abba Ari, and Amine Adouane. Distributed fault-tolerant algorithm for wireless sensor networks. *Int. J. Commun. Networks Inf. Secur.*, 9(2), 2017. URL: `http://www.ijcnis.org/index.php/ijcnis/article/view/1848`.

29 Lijun Wang, Jia Yan, Tao Han, and Dexiang Deng. On connectivity and energy efficiency for sleeping-schedule-based wireless sensor networks. *Sensors*, 19(9):2126, 2019. `doi:10.3390/s19092126`.

30 Rong Zheng and Robin Kravets. On-demand power management for ad hoc networks. *Ad Hoc Networks*, 3(1):51–68, 2005. `doi:10.1016/j.adhoc.2003.09.008`.

A An Example for a Colored Binary Tree in the O-LOCAL Algorithm

An example of a tree in the internal memory of each processor, for $q = 8$. A pair of neighbors u, v (not depicted) are colored by 9 and 13, respectively. In green are the rounds in which vertices that correspond to color 9 are awake. In orange are the rounds in which the vertices that correspond to color 13 are awake. The lowest common ancestor of these two colors is 12. In this round both u and v awake, and v receives the decision of u. (Note that both u and v are also awake in round 8, but in this round u may have not reached a decision yet, since its color is $9 > 8$.)



B Proof for Lemma 4.1

Proof. We prove the lemma by induction on the colors of the orientation.

Base: For the left-most leaf in the tree, the mapping of the first color in the orientation maps to the first awakening round of the algorithm. Vertices with the first colors of the orientation have no outgoing edges and need not wait for decisions of any of their neighbors. As they wake at the first round they make a decision to be in the MIS and sleep again.

Step: Let v be a vertex which awakes in round r' and assume by induction that all neighbors with lower colors already made a decision to be in the MIS or not. Let $\hat{\Delta}$ be the number of neighbors of v with colors smaller than the color of v. Let $S = s_1, \ldots, s_{\hat{\Delta}}$ be the rounds mapped to each color of these neighbors. Then we have $r' > \{s_i \in S\}$. Thus, in the binary tree, for each $i \in [\hat{\Delta}]$, r' and s_i have a lowest common ancestor with ID t, such that $s_i \ < t < r'$. (See Figure 1.) This is because a lowest common ancestor of two leaves must have these leaves in distinct subtrees rooted in its children. Otherwise, if both leaves belong to the same subtree of a child of a common ancestor, it is not the lowest one.

Let u be a neighbor of v with a color corresponding to the mapping s_i. We note that s_i must be in the subtree rooted in the left child of the ancestor of ID t and r' is in the subtree rooted in the right-child of the ancestor t. Both u and v are awakened in round t according to our algorithm. At round t, since $s_i < t$, the vertex u already made a decision if it is in the MIS or not, by the induction hypothesis. Thus, u sends a message with its decision to v at round t. Since $r' > t$, at round t, v simply receives the messages and awaits round r' to make a decision. (During this waiting period, the vertex v may communicate with additional neighbors.) When round r' finally arrives, all neighbors with lower colors, those in S, have made decisions and sent their decision in the round corresponding to some common ancestor with v in the binary tree. Thus, v has learnt the decisions of neighbors with smaller colors than its own. Finally, v makes a decision in round r' according to all the decisions made by neighbors in S. This concludes the proof of the lemma. ◀

C Psuedo-Code for DLT Construction

Algorithm 1 Connection(G).

```
/******** First Connection Stage ***********/
for each tree $T \in G$ in parallel do
    Scan all edges $(u, w) \in G$, such that $u \in T$ and $w \notin T$. Search for an edge $e = (u, w)$, such that the first coordinate in $L(w)$ is the smallest, and this coordinate is smaller than the ID of $T$.
    if such an edge $e$ was found then
        Connect $T$ to the tree to which $w$ belongs, and make that tree a parent of $T$.
        The vertex $u$ becomes the new root of $T$. Set the label of each vertex $v \in T$ to be $\langle L(T), l\rangle$ where $l = dist_T(u, v)$.
    else
        Mark $T$ as a local minimum tree.
    end if
end for
/******** Second Connection Stage ***********/
for each local minimum tree $T'_i$ in parallel do
    Perform a convergecast in $T'_i$ to check if $T'_i$ became a parent of another tree.
    if $T'_i$ did not became a parent of another tree then
        Choose an edge $e'$ with one endpoint in $T'_i$, connecting to another connected component arbitrarily.
    end if
end for
Wake up all vertices in $G$ for one round and exchange knowledge about all edges $e'$ that cross between components $T'_i$, $T'_j$, $i \neq j$. This is achieved by sending component labels over the edges $e'$ in parallel within one round. In the end of this round, all vertices enter the sleep state.
for each connected component $C$ in parallel do
    Using convergecast collect information about all vertices of $C$ in the root of $C$.
    The root $r_C$ of $C$ executes a BFS internally and reassigns labels to each vertex in $v \in C$, such that the new label is $\langle L(C), l\rangle$, where $l = dist_C(u, v)$. In this reassignment, the root $r_c$ remains the same.
    Broadcast the result to all vertices in $C$.
end for
```