Available at WWW.MATHEMATICSWEB.ORG
POWERED BY SCIENCE @ DIRECT®

Journal of Algorithms 46 (2003) 1–20

Journal of Algorithms

www.elsevier.com/locate/jalgor

Centralized broadcast in multihop radio networks ☆

Iris Gaber and Yishay Mansour *

Computer Science Department, Tel Aviv University, Tel Aviv, Israel

Received 2 July 1998

Abstract

We show that for any radio network there exists a schedule of a *broadcast* whose time is $O(D+\log^5 n)$, where D is the diameter and n is the number of nodes. (This result implies an optimal broadcast to networks with $D=\Omega(\log^5 n)$.) We present a centralized randomized polynomial time algorithm that given a network and a source, outputs a schedule for broadcasting the message from the source to the rest of the network.

Keywords: Radio networks; Broadcast; Algorithms

1. Introduction

The importance of radio communication networks is increasing. While in the past radio networks were used mainly for military applications today the focus shifts to civilian applications such as cellular phones and wireless local area networks (LAN). One of the main advantages of radio communication is the relative small investment in a rigid infrastructure. In addition, radio networks allow mobile users, a feature which is crucial in many applications. In this paper, however, we deal with multihop radio networks for which the topology is fixed.

One of the basic tasks in radio networks is broadcast. Broadcast is the dissemination of a message from a source node to all the nodes in the network. Broadcast is used for checking that a station exists (e.g., in a wireless LAN), propagating topological

☆ A preliminary version appeared in the 6th ACM–SIAM Symposium on Discrete Algorithms, 1995, pp. 577–585.

* Corresponding author.

E-mail addresses: gaber@server.mta.ac.il (I. Gaber), mansour@cs.tau.ac.il, mansour@math.tau.ac.il (Y. Mansour).

0196-6774/03/$ – see front matter © 2003 Elsevier Science (USA). All rights reserved.
doi:10.1016/S0196-6774(02)00292-4

information, etc. While such tasks exist in other types of networks, the lack of physical connection, and the user mobility, makes the broadcast a crucial task for radio networks. The transmission media of a radio network is simply the air, which is a shared media. As such it suffers from the collision effect, namely, when two neighboring nodes of a station transmit a message simultaneously, then neither of the messages is received at the station.

The design of broadcast protocols for a radio network is well studied and various algorithms have been proposed. We will review here some of the previous works, with an emphasis on works with a theoretical motivation and proven results. In order to compare the various results it would be helpful to use the parameters n for the number of nodes in the network and D for the diameter of the network.

The early works introduced distributed and centralized algorithms that compute a schedule for broadcast in a radio network [12,13] and their time complexity is at least the maximum node degree, which can be $\Omega(n)$, independent of D. The more recent work can be divided into the following categories:

(1) *Centralized* (each node needs to know the topology of the entire network). A centralized deterministic polynomial time algorithm for scheduling a bipartite graph in $O(\log^2 n)$ time is given in [14], where they also extend this result to a general network by dividing the graph into levels (according to their distance from the source) and treats each pair of consecutive levels as a bipartite graph, showing a schedule of $O(D \log^2 n)$ time. This is optimal for networks with constant diameter, due to the lower bound in [1].
(2) *Distributed* (only the knowledge of the topology local to each node is needed). These algorithms can be divided into:
   (a) *Deterministic protocols.* A general $\Omega(D \log n)$ lower bound is proven in [9], even for the case where $D = \Omega(n)$, while a trivial upper bound is $O(n^2)$. In a sequence of works [10,11,15,17] the upper bound has been improved to $O(n \log^2 n)$ [15]. Other deterministic protocols include [8,22].
   (b) *Randomized protocols.* The work of [7] gives a distributed randomized protocol which works in radio networks where a node has no topological knowledge of the network, not even about its neighbors. The expected time of the broadcast was improved to $O(D \log n + \log^2 n)$. This was proved to be optimal in [20] where it was shown that for any distributed algorithm, which does not depend on the topology of the network, there exists a graph for which the algorithm runs in expected time $\Omega(D \log(n/D))$.

In this work we seek bounds on the length of the optimal schedule. A lower bound of $\Omega(D)$ follows trivially, since the message needs to reach the furthest node from the source. We study the question whether for any graph there *exists* a schedule that runs in time $O(D)$. From [1] we know that this cannot be true in general, since there are graphs of $O(1)$ diameter that require $\Omega(\log^2 n)$ time.

We show that for any graph, with $D = \Omega(\log^5 n)$, there is a schedule of $O(D)$ time units. This resolves the open problem for almost the entire range of D. By [1], for $D = o(\log^2 n)$ there are graphs which require $\Omega(\log^2 n)$ time. We show that for $D = \Omega(\log^5 n)$, there is always a schedule of time $O(D)$. For the range of diameters between $\Omega(\log^2 n)$ and $O(\log^5 n)$ it remains an open problem whether there exists an $O(D)$ broadcast schedule.

Our result in constructive in the sense that we give a randomized algorithm that computes it in expected polynomial time. (For computing the schedule using a deterministic polynomial time algorithm we can derive a slightly worse result of $O(D + \log^6 n)$.)

In Section 2 we define the model and formalize the problem of broadcast. The overview of the algorithm is given in Section 3. In Section 4 we present the method of constructing the clusters with the desired properties. In Section 5 we define the clusters graph and build the clusters tree. The main algorithm including the specific protocols for the nodes is presented in Section 6. In Section 7 we prove the correctness and the time complexity of the schedule and in Section 8 we bound the computation time needed to construct the desired schedule. Finally, Appendix A describes the modification of the sparse partition which is used by the algorithm.

2. The model

We model a multi-hop radio network as an undirected graph $G(V, E)$, where an edge represents the ability of two nodes to directly communicate. We assume that the communication takes place in synchronous time-slots, this implies that all the nodes share a global clock, and messages are sent only at clock ticks.

We formalize the restriction of a radio network as follows: A processor receives a message at a certain time slot only if exactly one of its neighbors transmits at that time slot. (If more than one neighbor transmits, the processor does not receive any information.)

A *schedule* of a node defines for it in which time slots to transmit. A schedule of a network is composed from a schedule for each node.

The *Broadcast problem* is to send a single message from a given node, called *the source*, to all the other nodes in the graph. A schedule solves the broadcast problem if when the schedule terminates every node in the network has received the message.

The input to the scheduling algorithm is a connected graph $G(V, E)$ and a node $s \in V$ (the source), and the output is a schedule that solves the broadcast problem.

3. Algorithm overview

We wish to construct a schedule for broadcasting a message from the source node to the rest of the nodes in the graph. The intuition behind our algorithm is the following: We want the message to proceed “fast” to “distant” parts of the network. Once there is a copy of the message “near” each node, we can complete the broadcast easily.

Our algorithm begins by dividing the graph into x super-levels, according to their distance from the source, i.e., each super-level has D/x levels. (The value of the parameter x is determined through the analysis.)

Our work uses the idea of *network partition* [2,4,21] within each super-level. For each super-level we define clusters such that

(1) each cluster has a relatively small diameter,
(2) the union of the clusters covers the super-level, and
(3) the clusters graph can be colored by $O(\log n)$ colors. (The clusters graph is obtained by treating each cluster as a node, and introducing an edge between two nodes if in the original graph there is some edge that connects nodes from the corresponding clusters or if they share a common node.)

The objective is to send the message to a given node in each cluster. Once this is achieved, the other nodes in the cluster, which are close to such a recipient, obtain the message from it (using a broadcast protocol [7]). On the clusters' graph we build a directed spanning tree of depth x, where the directions are from lower to higher original BFS levels. Each node assigns a high priority to one of its sons and a low priority to the others. The priorities are assigned in such a way that a path from the source to a cluster contains at most x edges connecting a node with a low-priority son. Also, the message arrives at a high priority son after at most $O(D/x)$ time units, (since each path of the spanning tree of the cluster graph contains at most one cluster from each super-level). Thus the total delay of the high priority is $O(D)$, which accomplishes the task of "quickly" propagating the message to "distant" areas.

Having set up the graph in this manner, we can now build a schedule which is composed of three processes. These three processes are performed alternately, on different clock ticks.

During the first process, called *Broadcast-Through*, the message is propagated from the first node that receives the message within the cluster (referred to as *messenger*), to a unique node which we pick in advance (referred to as the *chosen representative*) that forwards the message to a cluster in the following super-level—the cluster with the highest priority. The path taken in this process consists of D/x nodes lying on consecutive BFS levels. The paths are assigned colors in such a way that (1) two paths of the same color cannot interfere with one another, and (2) paths with different colors can be pipelined so that they do not interfere with one another. The total time complexity of the Broadcast-Through process is $O(D + x\log n)$.

The second process is called *Broadcast-All*. Its purpose is to deliver the message from the messenger to all the nodes in its cluster. (The running time of the Broadcast-All is $O(D'\log n' + \log^2 n')$, where D' is the diameter of the cluster and n' is the number of nodes in the cluster.) After this the message is forwarded to all the neighboring clusters in the following super-level.

We prove that in order to reach any cluster C, the message is delayed by at most $O(\log n)$ Broadcast-All processes which occur in ancestor clusters of C. This is the main reason why the time of the Broadcast-All process is negligible in the overall cost, when the diameter is sufficiently large (i.e., $D = \Omega(\log^5 n)$ and $x = \log^3 n$).

The third process is called *Super-Level-to-Super-Level* and its purpose is to transfer the message between clusters of adjacent super-levels. Its time complexity is $O(\log^2 n)$.

4. Building the clusters

A central notion in our algorithm is the notion of a cluster. We divide the graph into clusters, which may be overlapping. Then, the broadcast is done on the graph of clusters.

Definition 1. We define all nodes with the same distance from the source as a *BFS level*, and we denote BFS level j by $l_j = \{v \mid \text{dist}(\text{source}, v) = j\}$.

Definition 2. We partition the graph to super-levels of D/x levels, where we denote super-level G_i by $G_i = \{v \mid v \in l_j,\ ((i-1) \cdot D/x) + 1 \leqslant j \leqslant i \cdot D/x\}$. Given the super-level G_i, the *lowest* level of G_i is $l_{((i-1)D/x)+1}$, and the *highest* level of G_i is $l_{i \cdot D/x}$. (Note that G_i is not necessarily connected.)

Our algorithm of building the clusters is applied to each super-level separately and consists of two stages:

(1) Defining pre-clusters in each super-level G_i. There exists one pre-cluster for each node of the lowest level of G_i. (This procedure is explained in Section 4.1.)
(2) Merging the pre-clusters into clusters that have the desired properties. (This procedure is explained in Section 4.2.)

4.1. Defining the pre-clusters

The pre-clusters of super-level G_i are defined as follows: Consider a super-level as a directed graph, in which the edges are all directed from lower to higher levels. In such a setting a pre-cluster $S_u^{(i)}$ includes all the nodes reachable from u.

This construction guarantees that each node at the highest level has an ancestor at the lowest level, such that any shortest path between them is fully contained within the pre-cluster.

4.2. Constructing the clusters

When building the clusters (as unions of pre-clusters) we have two major goals. The first is to have clusters with a diameter as small as possible, so that a message can be broadcast fast from one node to the rest of the nodes in the cluster. The second goal is to be able to color the clusters with a small number of colors. In clusters with the same color the broadcast can be performed in parallel. We like to minimize the number of coloring to increase the parallelism. Note that clusters might overlap, however two clusters that overlap are colored with different colors. The following theorem is a minor modification of Theorem 2.1 in [21]. (Appendix A contains the full proof of the theorem.)

Theorem 3. *For any graph $H(V, E)$ and any pre-clusters $S_1, \ldots, S_r$, $S_i \subset V$, such that the diameter of the subgraph induced by the nodes of S_i is at most ℓ, and $r \leqslant n$, there exist clusters $C_1, \ldots, C_k$ with the following properties*:

(1) *Each cluster C_j is a union of pre-clusters, i.e., $C_j = S_{j_1} \cup \cdots \cup S_{j_r}$.*
(2) *Each pre-cluster S_r is a member of at least one cluster C_j.*
(3) *The diameter of each cluster C_j is at most $\ell \cdot 2\log n$, and*
(4) *There is a coloring of the clusters C_j with $\lceil \log n \rceil$ colors.*

In addition, the clusters can be constructed from the pre-clusters in $O(n^3)$ time.

By applying the above theorem on the pre-clusters of a graph G_i, we get clusters $C_j^{(i)}$. Next we show a basic property of the clusters we create. Namely, that the cluster will contain, for any node at the highest level, a path to a node in the lowest level.

Definition 4. For two nodes, $u \in l_i$ and $v \in l_j$, $i < j$, let $\text{path}(u, v)$ be any shortest path between u and v in the graph in which each node is at the successive level of the previous one.

It follows from the definition that $\text{path}(u, v)$, for $u \in l_i$ and $v \in l_j$, consists of $j - i + 1$ nodes. Notice also that $\text{path}(u, v)$ does not always exist and it is not necessarily unique. Since a cluster is a union of pre-clusters we derive the following lemma.

Lemma 5. *Consider a super-level G_i, and a cluster $C_j^{(i)}$. For each node $v \in C_j^{(i)}$, at the highest level, i.e., $v \in l_{i \cdot D/x}$, there is a node $u \in C_j^{(i)}$, at the lowest level, i.e., $u \in l_{((i-1) \cdot D/x)+1}$, such that there exists a* $\text{path}(u, v)$ *for which* $\text{path}(u, v) \subseteq C_j^{(i)}$. (*We refer to such a u as the* ancestor *of v.*)

5. The clusters graph

After building the clusters in all the super-levels, we proceed to define the clusters graph. The nodes of the graph are the clusters, and two nodes are connected by an edge if the corresponding clusters have an edge connecting them or if there is a mutual node.

The general idea is to forward the message between the clusters of successive super-levels. Each cluster chooses a *sender*—a unique cluster in the preceding super-level—from which it will receive the message.

The procedure of picking the senders is applied to the clusters of each super-level, starting from the clusters of the last super-level, G_x, and continuing towards the clusters of the first super-level, G_1. Each cluster $C_j^{(i)}$ must first know all the clusters $C_r^{(i+1)}$ that have chosen it as their sender, before it decides which cluster $C_k^{(i-1)}$ to choose as its own sender. We define a subgraph of the clusters graph which is a directed tree where an edge from $C_j^{(i)}$ to $C_r^{(i+1)}$ indicates that $C_r^{(i+1)}$ chose $C_j^{(i)}$ as its sender.

Then, we start the processing of super-level G_i. For each cluster $C_r^{(i+1)}$, from super-level G_{i+1}, there is a unique cluster sender $C_j^{(i)}$ in which there is a unique node which is its *representative* in the highest level of G_i. This means that when we consider a cluster $C_j^{(i)}$, some of its nodes at the highest level may be chosen by clusters from super-level G_{i+1}. Those clusters would be called the *sons* of $C_j^{(i)}$. We apply the following series of steps to cluster $C_j^{(i)}$.

(1) The cluster $C_j^{(i)}$ determines its rank according to the rank of its sons (which are clusters in the following super-level that chose it as a sender) using the ranking procedure which is described in Section 5.1.

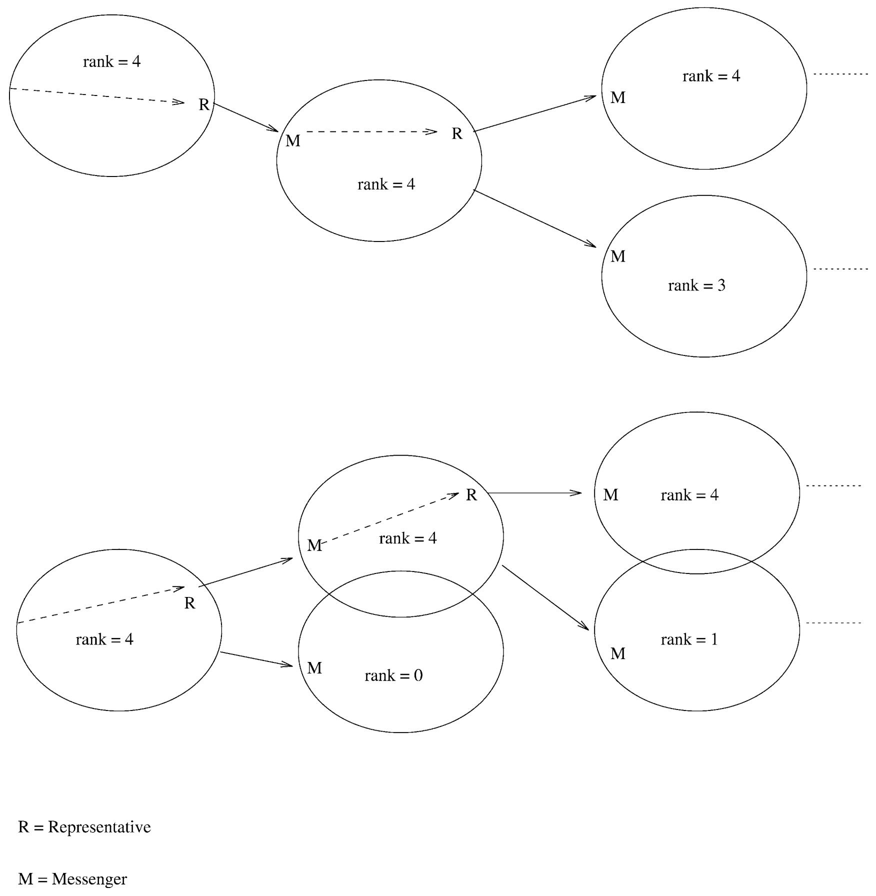

Fig. 1. The clusters graph.

(2) Using the ranking results of $C_j^{(i)}$ we chose a single node (from the nodes which were marked as representatives) and call it the *chosen representative*.
(3) Using the chosen representative v, we find the ancestor of u in $C_j^{(i)}$ (which is in the lowest level of G_i) and call it the *messenger*.
(4) We pick a node in G_{i-1} which is connected to the messenger node and call it the *representative*. (In the broadcast the representative will forward the message to its messenger.) The cluster of the representative would be the sender of $C_j^{(i)}$.

See Fig. 1 for an example of a clusters' graph. Note that the tree relation between the clusters depends heavily on the selection of the representatives and messengers.

5.1. The ranking procedure

Given the ranks of the sons of a cluster C, say $r_1, \ldots, r_k$, let $r_{\max} = \max_i\{r_i\}$. If there is a unique son whose rank is $r_{\max}$, then the rank of C is $r_{\max}$. Otherwise, there are at least two sons with rank $r_{\max}$, and in this case the rank of C is $r_{\max} + 1$. (If C does not have any sons its rank is 1.)

Since the edges between the representatives and the messengers create a tree structure between the clusters, we can easily bound the rank as a function of the number of nodes.

Lemma 6. *The rank of any cluster C is at most* $\log n$.

Note that this is the standard definition of a rank of a tree (see [16]).

5.2. Picking the messenger

For a cluster C in super-level G_i, if it has a son cluster F in G_{i+1}, such that $\text{rank}(C) = \text{rank}(F)$, the node $v \in C$ which is the representative of F in C will be called the *chosen representative*. (Note that if such an F exists it is unique, and therefore we select only one chosen representative.) The *messenger* of C will be the *ancestor* of the chosen representative. (By Lemma 5, such a node exists in C.)

If C has no such son cluster F (either it has no son cluster, or all its son clusters have a strictly smaller rank) the *messenger* will be chosen arbitrarily from the nodes at the lowest level.

5.3. Picking the representative and sender

When a messenger for a cluster C is chosen, it picks one of its neighbors at the previous level as the representative. Such a node exists—by the definition of the BFS levels. The messenger lies at the lowest level of its super-level, G_i, i.e., in level $l_{((i-1)\cdot D/x)+1}$. Therefore, its representative lies at the highest level of the previous super-level, G_{i-1}, i.e., in level $l_{(i-1)\cdot D/x}$. The sender of C is the cluster to which the representative belongs.

6. The main algorithm

The broadcasting starts at the sender which is part of the first super-level, G_1. G_1 consists of a single cluster, since the lowest level contains only one node, the source, from which all the other nodes of the super-level are reachable.

The message is then passed from clusters in G_i to clusters in G_{i+1} along the edges from the representatives of clusters in super-level G_i to the messengers of clusters in G_{i+1}. (These are the edges of the tree defined in the previous section.)

Each cluster $C_k^{(i)}$, passes the message to neighboring clusters $C_j^{(i+1)}$. The message advances in two ways within the cluster. A fast way, from the messenger to the chosen representative, along the path connecting them, and a slow way, where it is being broadcast to all the nodes in the cluster. Also there is a third way which is used to send the message between super-levels. The three ways are interleaved and do not interfere with each other.

6.1. *The procedure for a single cluster*

Each cluster, when its messenger receives the message, runs the following processes:

- *Broadcast-Through*: The message is passed from the messenger u to the chosen representative v along $\text{path}(u, v)$. Recall that the chosen representative is unique, and the path is fully contained within the cluster.
- *Broadcast-All*: The message is broadcast to all the nodes in the cluster, including the other representatives.
- *Super-Level-to-Super-Level*: A representative, upon receiving the message, broadcast it to the messenger (which is at the next super-level).

The three tasks are performed in parallel using time division. We refer to the different clock ticks of the clock as three different clocks. The *Broadcast-Through* clock is active at $t \equiv 0 \pmod 3$. The *Broadcast-All* clock is active at $t \equiv 1 \pmod 3$. The *Super-Level-to-Super-Level* clock is active at $t \equiv 2 \pmod 3$.

6.2. *Broadcast-Through process*

When the messenger of a cluster (whose color is j, $1 \leqslant j \leqslant \lceil \log n \rceil$) gets the message, it waits until the Broadcast-Through clock has time $\equiv 3 \cdot j \pmod{3 \cdot \lceil \log n \rceil}$ and then starts passing the message along the path to the chosen representative. Each node on that path, upon receiving the message, passes it on to the next node on the path, during the Broadcast-Through clock ticks. (We are working modulus $3 \cdot \lceil \log n \rceil$ since we like clusters of different colors to start at least three clock ticks apart. The reason why we need the separation of three is explained in the proof of Lemma 8.)

6.3. *Broadcast-All process*

The broadcasting to all the nodes of the cluster is done using the algorithm *Anonymous Broadcast*, which is based on the randomized algorithm of [7]. Technically, the *Anonymous Broadcast* run the randomized algorithm until it outputs a schedule of time $O(D \log n + \log^2 n)$. It is easy to see that the expected number of trials is $O(1)$. This observation leads to the following theorem.

Theorem 7. *Given a graph, $H(V, E)$, and a set of nodes $U \subset V$, all holding the same message, there exists a schedule for broadcasting to the remaining nodes in time $T(D, n) = K \cdot (D \log n + \log^2 n)$, where K is a constant, n is the number of nodes and D is the maximum distance from a node $v \in V$ to the nearest node $u \in U$, i.e., $D = \max_{v \in V} \min_{u \in U} \text{dist}(v, u)$. In addition there exists a randomized algorithm that computes the schedule in expected polynomial time.*

Note that the schedule that *Anonymous Broadcast* outputs is *always* of time $T(D, n)$, the expectation is only over the running time of the procedure. We can replace the randomized procedure by the deterministic algorithm of [14]. This will cause the schedule to be of time $O(D\log^2 n)$ and the overall schedule to be of time $O(D + \log^6 n)$.

When broadcasting to all the nodes of a cluster C, regardless of the color, the messenger v waits until the Broadcast-All clock has time $\equiv 0 \pmod{T((D/x)\log n, n)}$. At this time v, along with other nodes at that super-level, it applies the *Anonymous Broadcast* algorithm.

Let $D' = O((D/x)\cdot\log n')$ be the diameter of the cluster C and n' be the number of nodes in the cluster. Since $n' < n$ we have that $T(D', n') < T((D/x)\log n, n)$. This implies that after at most $O((D/x)\log^2 n)$ clock ticks all the representatives of C at level $l_{i\cdot D/x}$ receive the message.

6.4. *Super-Level-to-Super-Level process*

When the message gets to a representative, it waits until the Super-Level-to-Super-Level clock has time $\equiv 0 \pmod{K\cdot\lceil\log^2 n\rceil}$ and then, along with the rest of the representative nodes at that level, apply the *Anonymous Broadcast* process. After $O(\log^2 n)$ clock ticks the message is broadcast from the representatives at level $l_{i\cdot D/x}$ of super-level G_i to the messengers at level $l_{(i\cdot D/x)+1}$ of super-level G_{i+1}.

6.5. *Interaction between processes*

When a messenger gets the message, it initiates the Broadcast-Through process and the Broadcast-All process in its cluster. When a representative receives the message, it initiates the Super-Level-to-Super-Level process.

Any other node which gets the message acts according to the current context. On the ticks of the Broadcast-All clock, all nodes perform the *Anonymous Broadcast* algorithm. On the Broadcast-Through clock, a node on the path from a messenger to a chosen representative that receives the message, forwards the message to the next node on the path at the next Broadcast-Through clock tick. Note that a node may belong to a few clusters, but such clusters would have different colors, and therefore their schedule would not overlap.

7. Proof of correctness and complexity

We start by bounding the time it takes from the time the messenger of a cluster first receives the message, until the message arrives at all the cluster's nodes, and the chosen representative in particular.

The three processes: Broadcast-Through (for updating the chosen representative), Broadcast-All (for broadcasting to the rest of the cluster) and Super-Level-to-Super-Level (for forwarding to the next super-level) are performed at the different clocks and do not interfere one with the other. Therefore, we can examine the complexity of each one separately.

Given the time bounds for each of the processes, we proceed to investigate the time it takes for the message to get from one cluster representative, to the messenger of its son-cluster. Finally, we prove that every node gets the message after no longer than

$O(D + x \cdot \log^2 n + (D/x) \cdot \log^3 n)$ clock ticks. (Thus also proving the correctness of the protocol.)

Lemma 8. *Consider a node $v \in C$ of level ℓ that transmits at time t of the Broadcast-Through clock. Any other node $u \in C'$ of level ℓ that transmits in time t has* $\text{color}(C') = \text{color}(C)$. *Also, the next node w on the path to the chosen representative of C, receives the message at time t (i.e., the transmission to it does not collide).*

Proof. We need to consider only the Broadcast-Through processes, since the other processes run at different clock ticks. Assume the color of C is j and that the messenger starts broadcasting at time $t_0 = 3 \cdot j \pmod{3 \cdot \lceil \log n \rceil}$ of the Broadcast-Through clock. By definition, the path from the messenger to the chosen representative advances a level every Broadcast-Through clock tick. The message is broadcast from the kth level in G_i to the $k+1$ level at time $t_0 + k = 3j + k \pmod{3 \cdot \lceil \log n \rceil}$. Therefore at any time t and any level, there exists at most one color such that nodes belonging to clusters of that color might be sending a message.

By Lemma 5 the path to the chosen representative is fully contained within the cluster C, therefore $w \in C$. By the definition of the coloring, no node in C has an edge to a different cluster of the same color, in particular w does not have such a neighbor. Since v is the only node in C that is transmitting, the message to w does not collide with any other transmissions of nodes in clusters with color j.

Consider a node at the same super-level with a different color, j'. The Broadcast-Through process started either at least three Broadcast-Through clock tick before t_0 or at least three Broadcast-Through clock ticks after t_0. Since the message advances one level each clock tick, the difference between them remains three, and hence no collision will occur. In other words, when a node from a cluster whose color is j sends a message at level k, no node of a different color, in any of the levels between $k - 2$ and $k + 2$, sends a message at that time. (Here is where we use the separation of three clock ticks. If we had a separation of only two, a collision could occur if, for example, some node v at level k transmits to node w, at level $k + 1$, at the same time that some node u in level $k + 2$ transmits to node w', at level $k + 3$. If there is an edge (w, u) in G, then the transmission from v to w would collide, at w, with the transmission of u.) □

Lemma 9. *Suppose the messenger of a cluster $C_j^{(i)}$, from super-level G_i, gets the message at time t_j. The chosen representative of cluster $C_j^{(i)}$, receives the message by time $t_j + O((D/x) + \log n)$ and the rest of the nodes (including the other representatives) receive the message by time $t_j + T((D/x) \log n, n) = t_j + O((D/x) \log^2 n)$.*

Proof. Let α be the color of cluster $C_j^{(i)}$. The proof consists of two parts, one for the chosen representative, which uses the Broadcast-Through process, and the other for the other nodes, using the Broadcast-All process.

Broadcast-Through process: On the Broadcast-Through clock ticks, the messenger passes the message to the next node on the path to the chosen representative. First the messenger waits until the time becomes $3 \cdot \alpha \pmod{3 \cdot \lceil \log n \rceil}$ (which is at most $3 \cdot \lceil \log n \rceil$

clock ticks) and then the message is sent along the path to the representative. By Lemma 8 the message does not collide with any other transmission, and hence advances a level each Broadcast-Through clock tick. The path from the messenger to the chosen representative is of length D/x (as shown in Lemma 5) therefore the message reaches the chosen representative after D/x Broadcast-Through clock ticks. Consequently, the Broadcast-Through process take at most $O(\log n + D/x)$ clock ticks.

Broadcast-All process: On the Broadcast-All clock ticks we broadcast the message to all nodes of $C_j^{(i)}$. The messenger first waits until the time becomes 0 (mod $T((D/x)\log n, n)$) and then, along with other nodes at its level, broadcasts the message to all nodes at distance $D' \leqslant (D/x) \cdot 2\log n$ from it. All messengers that get the message at the same $T(D', n)$ time interval as the messenger of $C_j^{(i)}$, wait at most $T(D', n)$ Broadcast-All clock ticks before they start broadcasting, and then run together the *Anonymous Broadcast* algorithm. By Theorem 7 it follows that after at most $T(D', n)$ Broadcast-All clock ticks all nodes get the message and therefore two runs of the algorithm never interfere with one another. Consequently, the Broadcast-All passes take at most $T(D', n) = O((D/x)\log^2 n)$. □

Finally, we need to analyze the time it takes to transmit the message between super-levels.

Lemma 10. *Consider two super-levels G_i and G_{i+1} that contain the clusters $C_j^{(i)}$ and $C_k^{(i+1)}$, respectively, where $C_j^{(i)}$ is the sender of $C_k^{(i+1)}$. Let v_i be the representative of $C_j^{(i)}$ and u_{i+1} the messenger of $C_k^{(i+1)}$. Suppose v_i gets the message at time t_m. Then u_{i+1} receives it by time $t_m + O(\log^2(n))$.*

Proof. When v_i gets the message it waits until the time is 0 (mod $K \cdot \log^2(n)$) then it starts broadcasting using the *Anonymous Broadcast* algorithm.

By Theorem 7 it follows that after $K \cdot \log^2 n$ clock ticks all nodes get the message and therefore each invocation of the algorithm starts after the previous one has ended. Also, since this algorithm is performed in a separate clock it does not interfere with the other processes. Therefore, u_{i+1} gets the message after at most $2K \cdot \log^2 n$. □

Theorem 11. *A vertex waits no longer than $O(D + x\log^2 n + (D/x)\log^3 n)$ before it gets the message.*

Proof. Consider a node v in super-level G_i. There is at least one cluster $C_j^{(i)}$ such that $v \in C_j^{(i)}$. In the cluster's graph the relationship "sender of" defines a rooted tree. The path to node v includes all the clusters on the path from the source to the cluster $C_j^{(i)}$. Some of the clusters on the path are the *chosen son* of their father (i.e., their representative in the sender cluster is the chosen representative). In the broadcast, in order to be certain that the message has reached the chosen son it is sufficient that the Broadcast-Through protocol terminated, while for the other sons we need that the Broadcast-All process has terminated. In addition we have a Super-Level-to-Super-Level process each time the message advances to the next cluster.

From the fact that a rank can be no more than $\log n$ (Lemma 6) it follows that there are at most $\log n$ clusters on the path to $C_j^{(i)}$ which are not the chosen sons. This means that in at most $\log n$ places on the path we need to reach a representative which is not the chosen representative.

Lemma 9 proves that it takes at most $O(D/x + \log n)$ time to reach a chosen representative, and that it takes at most $O((D/x)\log^2 n)$ time to reach a representative which is not chosen. Lemma 10 proves that each Super-Level-to-Super-Level transfer takes $O(\log^2 n)$ time.

The path has i clusters, $i \leqslant x$, therefore the total time to reach any node v is bounded by

$$O\bigl(x\bigl((D/x) + \log n\bigr) + \log n\bigl((D/x)\log^2 n\bigr) + x\log^2 n\bigr),$$

which is

$$O\bigl(D + x \cdot \log^2 n + (D/x)\log^3 n\bigr). \qquad \square$$

Theorem 12. *A vertex waits no longer than $O(D + \log^5 n)$ before it gets the message.*

Proof. Let $x = \log^3 n$ in Theorem 11. $\square$

8. The sequential complexity of the algorithm

The preprocessing phase of the algorithm includes:

(1) computing the clusters,
(2) for each cluster selecting the chosen representative and messenger, and
(3) selecting the parent cluster (and thus implicitly defining the clusters tree).

Lemma 13. *The preprocessing phase takes $O(n^3)$ time, where n is the number of nodes.*

Proof. The algorithm begins by generating BFS levels: starting from the source which is labeled 0, it labels all neighbors of marked nodes with increasing numbers. The complexity of a BFS in a graph is $O(m + n)$, where m is the number of edges (see, e.g., [16]).

In the next stage we define the pre-clusters. Each node whose BFS number is a multiple of D/x becomes a kernel for a pre-cluster. The algorithm adds to the pre-cluster all the kernel's neighbors at distance less or equal to D/x that satisfy the following condition: the distance from the neighbor to the kernel is the same as the difference between their BFS numbers. When complete, such a pre-cluster induces a sub-graph whose edges connect nodes that lie on different BFS levels in the original graph. Note that the number of pre-clusters is bounded by n, and that we construct each pre-cluster in time $O(m)$. Therefore, the total complexity of this stage is $O(n \cdot m)$.

The construction of the clusters from the pre-clusters, by Theorem 3, takes $O(n^3)$.

The last procedure is that of assigning ranks to the clusters, and picking a sender for each cluster accordingly. For each cluster, we compute the rank based on the rank of its sons. The complexity of this task is $O(n)$. Thus the total time of the algorithm is $O(n^3)$. $\square$

Given the output of the preprocessing phase, we derive a schedule for each cluster (using Theorem 7) and establish the following theorem.

Theorem 14. *There exists a randomized polynomial time algorithm that for any graph G computes a schedule of length $O(D + \log^5 n)$.*

Appendix A. Constructing the clusters

Given the collection of pre-clusters, $S_1^{(i)}, \ldots, S_m^{(i)}$, we are going to build clusters $C_1^{(i)}, \ldots, C_k^{(i)}$ with the following properties:

(1) Each C_j is a union of some S_r's.
(2) Each S_r participates in some C_j.
(3) The radius of each cluster C_j is at most $(D/x) \cdot 2\log n$, i.e., $\forall u, v \in C_j^{(i)}$ $\operatorname{dist}(u, v) \leqslant D/x \cdot 2\log n$.
(4) There is an α coloring on the clusters C_j with $\alpha = \lceil \log n \rceil$ colors.
(5) The clusters are constructed in polynomial time.

The algorithm we use is similar to the algorithm *Max Cover* presented in [2], and we try to use here the same terminology. The major difference is that the output cover we generate, requires few colors rather than a low degree. We start with a few definitions.

Definition 15. A *cover* of the graph $G = (V, E)$ is a collection of clusters $\mathcal{S} = \{S_1, \ldots, S_m\}$ that contain all the vertices of the graph, i.e., such that $\bigcup \mathcal{S} = V$.

A *partial partition* of G is a collection of *disjoint* clusters $\mathcal{S} = \{S_1, \ldots, S_m\}$, i.e., with the property that $S \cap S' = \emptyset$ for every $S, S' \in \mathcal{S}$.

A *partial separated partition* of G is a partial partition such that for every $v_i \in S_i$ and $v_j \in S_j$ then $(v_i, v_j) \notin E$, i.e., there is no edge connecting two clusters.

Next we define a radius of a cluster and a set of clusters.

Definition 16. For a cluster S and a vertex $v \in S$, we define the radius of S w.r.t. v as in the induced subgraph $G(S)$, namely,

$$\operatorname{Rad}(v, \mathcal{S}) = \operatorname{Rad}\bigl(v, G(S)\bigr) = \max_{w \in S}\bigl(\operatorname{dist}_{G(S)}(v, w)\bigr)$$

and $\operatorname{Rad}(S)$ denotes the *radius* of S,

$$\operatorname{Rad}(S) = \min_{v \in S}\bigl\{\operatorname{Rad}(v, S)\bigr\}.$$

We denote the radius of a collection of clusters $\mathcal{S}$ as the maximum radius of a cluster $S \in \mathcal{S}$, i.e.,

$$\operatorname{Rad}(\mathcal{S}) = \max_{S \in \mathcal{S}}\bigl\{\operatorname{Rad}(S)\bigr\}.$$

We define the notion of one set system to coarsen another set system.

Definition 17. Given two collections of clusters, $\mathcal{S} = S_1, \ldots, S_p$ and $\mathcal{T} = T_1, \ldots, T_q$ we say that $\mathcal{T}$ *coarsens* $\mathcal{S}$ if the clusters of $\mathcal{S}$ are fully subsumed in those of $\mathcal{T}$, i.e., for every $S_i \in \mathcal{S}$ there exists $T_j \in \mathcal{T}$ a such that $S_i \subseteq T_j$.

Lastly we define a neighbor relationship between clusters.

Definition 18. A cluster F is defined to be a *neighbor* of a cluster U if there exist nodes $u \in U$ and $f \in F$ such that $(u, f) \in E$.

A.1. Procedure Part-Sep

In this section we present a procedure which will be used as a building block in the construction of the new collection of clusters. The input is a graph G and a collection of pre-clusters, $\mathcal{R}$. The output consists of a partial separated partition, i.e., a collection of clusters, $\mathcal{DT}$, that subsume a subset $\mathcal{DR} \subseteq \mathcal{R}$ of the original pre-clusters. The goal is to subsume "many" pre-clusters of $\mathcal{R}$ while maintaining the radii of the output clusters in $\mathcal{DT}$ relatively small.

The procedure starts by setting $\mathcal{U}$, the collection of *unprocessed* pre-clusters, to equal $\mathcal{R}$. The procedure operates in iterations. Each iteration constructs one output cluster $Z \in \mathcal{T}$, by merging together some pre-clusters of $\mathcal{R}$. The iteration begins by arbitrarily picking a pre-cluster $S_0 \in \mathcal{U}$ and designating it as the kernel of a cluster to be constructed next. The cluster is then repeatedly merged with intersecting and neighboring pre-clusters from $\mathcal{U}$. This is done in a layered fashion, adding one layer at a time. The pre-clusters of $\mathcal{U}$ added to Z are subsequently discarded from $\mathcal{U}$. At each stage, the original process is carried repeatedly until the next iteration increases the number of original pre-clusters merged into Z by a factor of less than 2. (This guarantees that there are at most $\log n$ iterations.) At that

```
$\mathcal{U} \leftarrow \mathcal{R}$; $\mathcal{DT} \leftarrow \emptyset$; $\mathcal{DR} \leftarrow \emptyset$
while $\mathcal{U} \neq \emptyset$ do
    Select an arbitrary cluster $S_0 \in \mathcal{U}$
    $\mathcal{Z} \leftarrow \{S_0\}$; $Z \leftarrow S_0$
    repeat
        $\mathcal{Y} \leftarrow \mathcal{Z}$; $Y \leftarrow Z$
        $\mathcal{Z} \leftarrow \{S \mid S \in \mathcal{U},\ S \in \text{neighbors}(Y)\}$.
        $Z \leftarrow \bigcup_{S \in \mathcal{Z}} S$
    until $|\mathcal{Z}| \leqslant 2|\mathcal{Y}|$
    $\mathcal{U} \leftarrow \mathcal{U} - \mathcal{Z}$
    $\mathcal{DT} \leftarrow \mathcal{DT} \cup \{Y\}$
    $\mathcal{DR} \leftarrow \mathcal{DR} \cup \mathcal{Y}$
end-while
Output $(\mathcal{DR}, \mathcal{DT})$.
```

Fig. A.1. Procedure Part-Sep.

point, the resulting cluster from the *previous* stage is added to the collection $\mathcal{DT}$. These iterations proceed until $\mathcal{U}$ is exhausted.

Note that each of the original pre-clusters in $\mathcal{DR}$ is covered by some cluster in $\mathcal{DT}$ constructed during the execution of the procedure. However, some original $\mathcal{R}$ pre-clusters are thrown out of consideration without being subsumed by any cluster in $\mathcal{DT}$. This is why a single application of the procedure in not enough, and many phases are necessary. A description of the procedure is given in Fig. A.1.

A.2. Analysis

The properties of the procedure are summarized by the following lemma.

Lemma 19. *Given a graph $G = (V, E)$, $|V| = n$ and a collection of pre-clusters $\mathcal{R}$, the collections $\mathcal{DT}$ and $\mathcal{DR}$ constructed by procedure Part-Sep satisfy the following properties*:

(1) *$\mathcal{DT}$ coarsens $\mathcal{DR}$.*
(2) *$\mathcal{DT}$ is a partial separated partition.*
(3) $|\mathcal{DR}| \geqslant |\mathcal{R}|/2$.
(4) $\mathrm{Rad}(\mathcal{DT}) \leqslant 2\log(|\mathcal{R}|) \cdot \mathrm{Rad}(\mathcal{R})$.

Proof. The construction of $\mathcal{DT}$ is done by adding a new cluster, $\{Y\}$, to it at each iteration. Since the cluster $\{Y\}$ that is being added is the cluster Z of the previous stage, and that Z was built from union of pre-clusters from $\mathcal{U}$ property (1) holds.

Property (2) is proved by showing that the collection of clusters Y are mutually disjoint and are not connected. Suppose, seeking to establish a contradiction, that there are two vertices v_i, v_j such that $v_i \in Y_i$ and $v_j \in Y_j$ and there is an edge (v_i, v_j) in E. Without loss of generality suppose that Y_i was created in an earlier iteration than Y_j. Since $v_j \in Y_j$ there must be a pre-cluster U_j such that $v_j \in U_j$ and U_j was still in $\mathcal{U}$ when the algorithm started constructing Y_j. But every such pre-cluster U_j is a *neighbor* of Y_i , and therefore the final construction step creating the collection $\mathcal{Z}$ from Y_i should have added U_j into $\mathcal{Z}$ and eliminating it from $\mathcal{U}$, a contradiction. Therefore the collection $\mathcal{DT}$ is a partial separated partition and property (2) is proved.

Property (3) is now derived as follows. It is immediate from the termination condition of the internal loop that the resulting pair $\mathcal{Y}, \mathcal{Z}$ satisfies $|\mathcal{Z}| \leqslant 2|\mathcal{Y}|$. Therefore, using the fact that the $\mathcal{Y}$ clusters are disjoint that has just been proved,

$$|\mathcal{R}| = \sum_{Z} |\mathcal{Z}| \leqslant \sum_{Y} 2|\mathcal{Y}| = 2|\mathcal{DR}|.$$

Finally, we analyze the increase in the radius of the clusters of $\mathcal{DT}$. Consider some iteration of the main loop of the algorithm, starting with the selection of some pre-cluster $S \in \mathcal{U}$. Let J denote the number of times the internal loop was executed. Denote the initial collection $\mathcal{Z}$ by $\mathcal{Z}_0$. Denote the collection $\mathcal{Z}$ constructed on the ith internal iteration ($1 \leqslant i \leqslant J$) by $\mathcal{Z}_i$. Note that $Y_i = Z_{i-1}$, Z_i is constructed on the basis of Y_i, and $Z_i = \bigcup_{S \in \mathcal{Z}_i} \mathcal{S}$. We show by the following two claims:

(1) $|\mathcal{Z}_i| \geqslant 2^i$ for every $0 \leqslant i \leqslant J-1$.
(2) For every $0 \leqslant i \leqslant J$, $\text{Rad}(Y_i) \leqslant 2i \cdot \text{Rad}(\mathcal{U})$.

Claim (1) hold since each time the termination condition of the internal loop is not valid, the size of Z at least doubles. Claim (2) follows since we increase in each internal loop the radius by at most $1 + \text{Rad}(\mathcal{R}) \leqslant 2\text{Rad}(\mathcal{R})$.

It follows from (1) that $J \leqslant \log(|\mathcal{R}|)$, and hence by (2) we have that $\text{Rad}(Y_j) \leqslant 2\log(|\mathcal{R}|) \cdot \text{Rad}(\mathcal{U})$, which completes the proof of property (4) of the theorem. □

A.3. Procedure cover

We now turn to construct the desired clusters. The algorithm involve multiple applications of the procedure Part-Sep from the previous section. The input to the algorithm is a graph $G = (V, E)$, $|V| = n$ and a cover $\mathcal{S}$. The output collection of clusters, $\mathcal{T}$, is initially empty. The algorithm maintains the collection of "remaining" pre-clusters $\mathcal{R}$. These are the pre-clusters not yet subsumed by the constructed collection. Initially, $\mathcal{R} = \mathcal{S}$, and the algorithm terminates once $\mathcal{R} = \emptyset$. The algorithm operates in at most $\lceil \log(|S|) \rceil$ phases. Each phase consists of the activation of the procedure Part-Sep, which constructs a partial separated partition $\mathcal{DT}$ coarsening some of the pre-clusters of $\mathcal{R}$. The algorithm then adds this subcollection of output clusters to $\mathcal{T}$, and removes the set of subsumed original pre-clusters $\mathcal{DR}$ from $\mathcal{R}$.

The fact that the collection of clusters $\mathcal{DT}$ constructed by the procedure Part-Sep is a partial separated partition ensures that the clusters of each phase can be colored with the same color. The formal description is given in Fig. A.2.

A.4. Analysis

Theorem 20. *Given a graph $G = (V, E)$, and a cover $\mathcal{S}$, the algorithm constructs a cover $\mathcal{T}$ that satisfies the following properties*:

(1) *$\mathcal{T}$ coarsens $\mathcal{S}$.*
(2) $\text{Rad}(\mathcal{T}) \leqslant 2\log(|\mathcal{S}|) \cdot \text{Rad}(\mathcal{S})$.
(3) *There is an α coloring on the clusters of $\mathcal{T}$ with $\alpha \leqslant \lceil \log(|\mathcal{S}|) \rceil$ colors.*

```
Set $i \leftarrow 1$
$\mathcal{R} \leftarrow \mathcal{S}$; $\mathcal{T} \leftarrow \emptyset$
**repeat**
  $(\mathcal{DR}, \mathcal{DT}) \leftarrow$ **Part − Sep**$(\mathcal{R})$
  $\mathcal{T} \leftarrow \mathcal{T} \cup \mathcal{DT}$
  Color the clusters of $\mathcal{DT}$ by color $i$
  $\mathcal{R} \leftarrow \mathcal{R} \setminus \mathcal{DR}$
  $i = i + 1$
**until** $\mathcal{R} = \emptyset$
```

Fig. A.2. Procedure cover.

Proof. Let $\mathcal{R}^i$ denote the contents of the collection $\mathcal{R}$ at the beginning of phase i, and let $r_i = |\mathcal{R}^i|$. Let $\mathcal{DT}^i$ denote the collection $\mathcal{DT}$ added to $\mathcal{T}$ at the end of phase i, and let $\mathcal{DR}^i$ be the collection $\mathcal{DR}$ removed from $\mathcal{R}$ at the end of phase i. The fact that $\mathcal{T}$ coarsens $\mathcal{S}$ (property (1)) follows from the fact that $\mathcal{T} = \bigcup_i \mathcal{DT}^i$, $\mathcal{S} = \bigcup_i \mathcal{DR}^i$ and by property (1) of Lemma 19, $\mathcal{DT}^i$ coarsens $\mathcal{DR}^i$ for every i. Property (2) follows directly from property (4) of Lemma 19. It remains to prove property (3). By property (2) of Lemma 19 it follows that each phase constructs a collection of clusters which are 2-separated, i.e., for each two clusters, C_p, C_q generated at the same phase $C_p \cap C_q = \emptyset$ and for each $v_p \in C_p$, $v_q \in C_q$ there is no edge $(v_p, v_q) \in E$. By the definition of coloring we can color all clusters generated at the same phase with a single color. Therefore it remains to bound the number of phases performed by the algorithm. This bound relies on the following observations. By property (3) of Lemma 19, in every phase, i, at least $|\mathcal{DR}^i| \geqslant |\mathcal{R}^i|/2$ pre-clusters of $\mathcal{R}^i$ are removed from the collection $\mathcal{R}^i$, i.e., $r_{i+1} \leqslant r_i/2$.

Consequently, since $r_0 = |\mathcal{S}|$, $\mathcal{S}$ is exhausted after no more than $\lceil \log(|\mathcal{S}|) \rceil$ phases of the algorithm, and hence the number of colors is no more than $\lceil \log(|\mathcal{S}|) \rceil$. This completes the proof of the theorem. □

A.5. Applying the algorithm

We can now use the algorithm presented here in order to construct the clusters from the original pre-clusters. The graph G we are using is the graph induced on the nodes of the super-level G_i and n will be the number of its nodes.

Notice the radius of each original pre-cluster is D/x and that the collection of the pre-clusters, $S_1^{(i)}, \ldots, S_m^{(i)}$ is a *cover* of the *super-level*, G_i. The input to the algorithm, is therefore: The graph, G_i, and the cover $\mathcal{S}$, i.e., the collection $S_1^{(i)}, \ldots, S_m^{(i)}$.

Theorem 21. *The output collection of clusters,* $C_1^{(i)}, \ldots, C_r^{(i)}$ *satisfies*:

(1) *Each C_j is a union of some S_r's.*
(2) *Each S_r participates in some C_j.*
(3) *The radius of each cluster C_j is at most* $2 \cdot D/x \cdot \log(n)$, *i.e.*, $\forall u, v \in C_j^{(i)}$ $\operatorname{dist}(u, v) \leqslant D/x \cdot 2\log(n)$.
(4) *There is an α coloring on the clusters C_j with $\alpha = \lceil \log n \rceil$ colors.*
(5) *The algorithm runs in time $O(m \log n)$.*

Proof. Property (1) follows directly from property (1) of Theorem 20 as coarsening means that every cluster is a union of some pre-clusters. Property (2) follows from the construction since every pre-cluster is taken out of $\mathcal{S}$ only when it joins a cluster. The radius of the pre-clusters is D/x, therefore we get from property (2) of Theorem 20 that the radius of the clusters is at most $2 \cdot \log n \cdot D/x$. That proves property (3). Finally, from property (3) of Theorem 20 we get that the coloring number of the clusters is less or equal to $\lceil \log(|\mathcal{S}|) \rceil$. Since $|\mathcal{S}| < n$, we have that α is less or equal to $\lceil \log n \rceil$. That concludes the proof of property (4).

As for complexity, the algorithm consists of $O(\log n)$ iterations of a procedure named Part-Sep. This procedure also consists of iterations, during each of which it takes a pre-cluster not yet processed to become a kernel for a new cluster. For each such kernel, it goes over its neighboring clusters. It can be seen that in this process no edge is checked more than once—after which it may merge another pre-cluster into a cluster. Whether it does so or not, the vertex whose pre-cluster is already in a cluster does not check its edges again. This procedure therefore takes $O(m)$ time and the total time of this stage is $O(m \log n)$. That concludes the proof of property (5). $\square$

References

[1] N. Alon, A. Bar-Noy, N. Linial, D. Peleg, A lower bound for radio broadcast, J. Comput. System Sci. 43 (2) (1991) 290–298.

[2] B. Awerbuch, D. Peleg, Sparse partitions, in: Proceedings of the 31st Annual Symposium on Foundations of Computer Science, Vol. 2, St. Louis, MO, 22–24, 1990, pp. 503–513.

[3] B. Awerbuch, D. Peleg, Synchronization with polylogarithmic overload, in: Proceedings of the 31st Annual Symposium on Foundations of Computer Science, 1990, pp. 514–522.

[4] B. Awerbuch, D. Peleg, Routing with polynomial communication-space trade-off, SIAM J. Discrete Math. 5 (1992) 151–162.

[5] B. Awerbuch, D. Peleg, Online tracking of mobile users, J. ACM 42 (1995) 1021–1058.

[6] Y. Afek, M. Ricklin, Sparser: A paradigm for running distributed algorithms, J. Algorithms 14 (1993) 316–328.

[7] R. Bar-Yehuda, O. Goldreich, A. Itai, On the time-complexity of broadcast in radio networks: An exponential gap between determinism and randomization, J. Comput. System Sci. 45 (1) (1992) 104–126.

[8] S. Basagni, D. Bruschi, I. Chlamtac, A mobility-transparent deterministic broadcast mechanism for ad hoc networks, IEEE/ACM Trans. Network. 7 (6) (1999) 799–807.

[9] D. Bruschi, M. Del Pinto, Lower bounds for the broadcast problem in mobile radio networks, Distrib. Comput. 10 (3) (1997) 129–135.

[10] B.S. Chlebus, L. Gasieniec, A. Ostlin, J.M. Robson, Deterministic radio broadcasting, in: Proc. 27th International Colloquium on Automata, Languages and Programming, ICALP', 2000, pp. 717–728.

[11] B.S. Chelbus, L. Gasieniec, A.M. Gibbons, A. Pelc, W. Rytter, Deterministic broadcasting in unknown radio networks, in: Proc. 11th Ann. ACM–SIAM Symposium on Discrete Algorithms, San-Francisco, CA, 2000, pp. 861–870.

[12] I. Chlamtac, S. Kutten, An efficient broadcast protocol embedded in multi-hop radio networks, in: COMPCON Conf. Proceedings, Fall, 1984, pp. 220–229.

[13] I. Chlamtac, S. Kutten, On broadcasting in radio networks-problem analysis and protocol design, IEEE Trans. Commun. 33 (12) (1985) 1240–1246.

[14] I. Chlamtac, O. Weinstein, The wave expansion approach to broadcasting in multihop radio networks, IEEE Trans. Commun. 39 (9) (1991) 426–433.

[15] M. Chrobak, L. Gasieniec, W. Rytter, Fast broadcasting and gossiping in radio networks, in: FOCS, 2000, pp. 575–581.

[16] T.H. Cormen, C.E. Leiserson, R.L. Rivest, Introduction to Algorithms, MIT Press, 1990.

[17] G. De Marco, A. Pelc, Faster broadcasting in unknown radio networks, Inform. Process. Lett. 79 (2) (2001) 53–56.

[18] I. Gaber, Y. Mansour, Broadcast in radio networks, in: The 6th ACM–SIAM Symposium on Discrete Algorithms, 1995, pp. 577–585.

[19] F.K. Hwang, The time complexity of deterministic broadcast radio networks, Discrete Appl. Math. 60 (1–3) (1995) 219–222.

[20] E. Kushilevitz, Y. Mansour, An $\Omega(D \log n)$ lower bound for broadcast in radio networks, SIAM J. Comput. 27 (3) (1998) 702–712.

[21] N. Linial, M. Saks, Low diameter graph decomposition, networks, Combinatorica 13 (4) (1993) 441–454.
[22] E. Pagani, G.P. Rossi, Reliable broadcast in mobile multi hop packet networks, in: Proceedings of the Third Annual ACM/IEEE International Conference on Mobile Computing and Networking (MobiCom'97), Hungary, 1997, pp. 34–42.