An improved algorithm for radio broadcast

(Extended abstract)

Michael Elkin [*†]   Guy Kortsarz[‡]

September 24, 2004

Abstract

We show that for every radio network $G = (V, E)$ and source $s \in V$, there exists a radio broadcast schedule for G of length $Rad(G,s) + O(\sqrt{Rad(G,s)} \cdot \log^2 n) = O(Rad(G,s) + \log^4 n)$, where $Rad(G,s)$ is the radius of the radio network G with respect to the source s. This result improves the previously best-known upper bound of $O(Rad(G,s) + \log^5 n)$ due to Gaber and Mansour [12].

For graphs with small genus, particularly for *planar* graphs, we provide an even better upper bound of $Rad(G,S) + O(\sqrt{Rad(G,s)} \cdot \log n + \log^3 n) = O(Rad(G,s) + \log^3 n)$.

[*]Department of Computer Science, Yale University, New Haven, CT, USA, 06520-8285, `elkin@cs.yale.edu`.

[†]This work was supported by the DoD University Research Initiative (URI) administered by the Office of Naval Research under Grant N00014-01-1-0795.

[‡]Computer Science department, Rutgers University, Camden, NJ, USA. Email: `guyk@crab.rutgers.edu`

1 Introduction

1.1 Definition and motivation

Consider a synchronous network of processors that communicate by transmitting messages to their neighbors, where a processor receives a message in a given step if and only if precisely one of its neighbors transmit. The instance of the radio broadcast problem, called *radio network*, is a pair $(G = (V, E), s)$, $s \in V$, where G is an unweighted undirected n-vertex graph, and s is a vertex, called *source*. The objective is to deliver one single message that the source s generates to all the vertices of the graph G using the smallest possible number of communication rounds. The prescription that tells each vertex when it should broadcast is called *schedule*; the *length* of the schedule is the number of rounds it uses, and it is called *admissible* if it informs all the vertices of the graph (see Section 2 for a formal definition).

From practical perspective, the interest to radio networks is usually motivated by their military significance, as well as by the growing importance of cellular and wireless communication (see, e.g., [17, 12, 4]). The radio broadcast is perhaps the most important communication primitive in radio networks, and, consequently, it has been intensively studied starting from mid-eighties [10, 14, 15, 8, 7, 9, 13, 12, 17, 1, 3, 4, 6, 5].

From theoretical perspective, the study of the radio broadcast problem provided researchers with a particularly convenient playground for the study of such broad and fundamental complexity-theoretic issues as the power and limitations of randomization, and of different models of distributed computation [4, 17, 15].

1.2 History and our results

One of the most important parameters of an instance (G, s) of the radio broadcast problem is its *radius*, denoted $Rad(G, s)$ or shortly Rad, and defined as the maximum distance $d_G(s, v)$ in the graph G between the source s and some vertex $v \in V$. It is easy to see that any admissible schedule for (G, s) must have at least $Rad(G, s)$ rounds.

A basic open question in this area is to understand how large can be the minimum length of an admissible schedule for an instance (G, s) in terms of $Rad(G, s)$ and n. Chlamtac and Weinstein [6] proved the first upper bound of $O(Rad \cdot \log^2 n)$, which was improved to $O(Rad \cdot \log n + \log^2 n)$ soon afterwards by Bar-Yehuda et al. [4]. Very recently a *deterministic and centralized* algorithm with the same number of rounds was devised by Kowalski and Pelc [16]. Alon et al. [1] have shown a lower bound of $Rad + \Omega(\log^2 n)$. Finally, on SODA 1995 Gaber and Mansour have published an upper bound of $O(Rad + \log^5 n)$, which is the state-of-the-art. See Table 1 for the summary of previous and our results.

Reference	Type of Bound	The bound
[6]	Upper	$O(Rad \cdot \log^2 n)$
[4, 16]	Upper	$O(Rad \cdot \log n + \log^2 n)$
[1]	Lower	$Rad + \Omega(\log^2 n)$
[12]	Upper	$O(Rad + \log^5 n)$
This paper	Upper (general graphs)	$Rad + O(\sqrt{Rad} \cdot \log^2 n) =$ $= O(Rad + \log^4 n)$
This paper	Upper (planar graphs)	$Rad + O(\sqrt{Rad} \cdot \log n + \log^3 n) =$ $= O(Rad + \log^3 n)$

Table 1: The summary of previous and our results.

Despite the recent intensive research of the radio broadcast [10, 14, 15, 8, 7, 9, 13], no progress has been made in narrowing the gap between the upper bound of $O(Rad + \log^5 n)$ of Gaber and Mansour [12] and the lower bound of $Rad + \Omega(\log^2 n)$ of Alon et al. [1] during the last decade. In this paper we improve the upper bound and devise an algorithm that returns schedules of length $Rad + O(\sqrt{Rad} \cdot \log^2 n) = O(Rad + \log^4 n)$ for general graphs. For graphs with small genus g our algorithm returns schedules of length $Rad + O(\sqrt{Rad} \cdot g^{1/4} \cdot \log n + \log^3 n)$. Particularly, for *planar* graphs the bound is $Rad + O(\sqrt{Rad} \cdot \log n + \log^3 n) = O(Rad + \log^3 n)$. The upper bound of $O(Rad + \log^3 n)$ applies actually for a more general family of graphs with genus $g = O(\log^4 n)$.

Remark: We believe that it is important to provide an upper bound of the form $Rad + O(\text{polylog}(n))$, and not $O(Rad + \text{polylog}(n))$, and it is significantly harder to prove a bound of the former type. Note that an upper bound of form $Rad + O(\text{polylog}(n))$ does not just improve the constant hidden by the O-notation, but rather shows that this constant is *precisely equal to 1.* In our opinion this sheds a new light on the behavior of the minimum length of the admissible schedule, in addition to the tighter bound on the degree of the logarithm in the upper bound. In addition, an upper bound of the form $Rad + O(\text{polylog}(n))$ can be seen as an *additive approximation algorithm* for the radio broadcast problem. (See also [11] for the corresponding lower bound.)

1.3 Proof techniques

Our algorithm for constructing a radio broadcast schedule of length $Rad + O(\sqrt{Rad} \cdot \log^2 n)$ is based on the algorithm of Gaber and Mansour [12]. Both algorithms partition the graph into *layers* (or, *super-levels*, cast to the terminology of [12]), and construct an $(O(\log n), O(\log n))$-partition of every layer (see Sec. 2.6 for the definition) using the algorithm of [18] or [2]. One difference between the algorithms is that in our algorithm the layers are *overlapping* and not *disjoint* as in the algorithm of [12]. This difference, while complicating the analysis, enables us to reduce the additive term from

$O(\log^5 n)$ to $O(\log^4 n)$. Both algorithms use the constructed partitions to control the data flow, and use similar procedures for this purpose. In [12] the scheduling of these procedures involves round-robin time sharing, which guarantees that executions of different procedures do not cause collisions. While very convenient for the analysis, this technique carries a price of increasing the length of the constructed schedule by a constant multiplicative factor. Our algorithm is geared to avoid this unnecessary multiplicative factor in order to achieve a *purely additive* approximation of the optimal schedule. To this end, our algorithm carefully combines different building blocks in such a manner that they cause no collision even though they employ *the same rounds*.

2 Preliminaries

We start with introducing some definitions and notations.

Definition 2.1 *The set of neighbors of a vertex v in an unweighted undirected graph $G(V, E)$, denoted $\Gamma_G(v)$, is the set $\{u \in V \mid (v, u) \in E\}$. For a subset $X \subseteq V$, the set of neighbors of the vertex v in the subset X, denoted $\Gamma_G(v, X)$ (or $\Gamma(v, X)$ when the graph G is clear from the context), is the set $\{u \in X \mid (v, u) \in E\}$.*

Notation 2.2 *1. For a positive integer number n, let $[n]$ denote the set $\{1, 2, \ldots, n\}$.*

2. For an index $i \in \{1, 2\}$, let $\bar{i}$ denote the complimentary index $2 - i$.

We also use the letter ϵ to denote an arbitrarily small positive real universal constant.

Definition 2.3 *Let $G = (V, E)$ be an unweighted undirected graph, and $R \subseteq V$ be a subset of vertices. The set of vertices* informed *by R, denoted $I(R)$, is $I(R) = \{v \mid \exists! x \in R \;\; s.t. \;\; v \in \Gamma_G(x)\}$ (the notation $\exists! x$ stands for "there exists a unique x"). For a singleton set $R = \{x\}$, $I(R) = I(\{x\}) = I(x) = \Gamma_G(x)$.*

A sequence of vertex sets $\Pi = (R_1, R_2, \ldots, R_q)$, $q = 1, 2, \ldots$, is called a *radio broadcast schedule* (henceforth referred as a *schedule*) if $R_{i+1} \subseteq \bigcup_{j=1}^{i} I(R_j)$ for every $i = 1, 2, \ldots, q - 1$. Intuitively, this condition means that the vertices that send a message in certain round have to be informed in one of the previous rounds.

The set of vertices *informed by a schedule* Π, denoted $I(\Pi)$, is $I(\Pi) = \bigcup_{R \in \Pi} I(R)$.

Given a graph $G = (V, E)$ and a vertex $s \in V$, a schedule Π is *admissible* with respect to (G, s) if $R_1 = \{s\}$ and $V \subseteq I(\Pi)$.

The *length* of the schedule $\Pi = (R_1, R_2, \ldots, R_q)$ is $|\Pi| = q$.

An instance of the *radio broadcast problem* $\mathcal{G}$ is a pair $(\bar{G} = (\bar{V}, \bar{E}), s)$, where $\bar{G}$ is a graph, and $s \in \bar{V}$ is a vertex. The goal is to compute an admissible schedule Π of minimal length. The *value* of an instance $\mathcal{G}$ of the radio broadcast problem is the length of the shortest admissible schedule Π for this instance.

For any schedule $\Pi = (R_1, R_2, \ldots, R_q)$, the set R_i is called the *ith round* of Π, $i = 1, 2, \ldots, q$. If a vertex v belongs to the set R_i, we say that v is *active* in round i.

Definition 2.4 *For a rooted tree $(T = (V_T, E_T), s)$, $s \in V_T$, the* parent *of a vertex $v \in V_T$, $v \neq s$, is the unique vertex u such that $(u, v) \in E_T$, and u belongs to the path between s and v in T. The vertex u is denoted $parent_T(u)$. The vertex v is also said to be a* child *of the vertex u in the rooted tree (T, s).*

For a rooted tree (T, s), the rank *of a vertex $v \in V_T$ is defined as follows. If v is a leaf, its rank is 0. For an internal vertex v, let R be the set of ranks of all the children of the vertex v, and let r be the maximum value in R. If there is exactly one child w of v with rank r, then rank of the vertex v is defined to be r; otherwise, it is defined to be $r + 1$.*

For a vertex $v \in V_T$, let $\ell_T(v)$ (or $\ell(v)$, when the tree T can be deduced from the context) denote the level *of the vertex v in the rooted tree (T, s), that is, the distance between the vertices s and v in T.*

The radius *of a rooted tree (T, s), denoted $Rad(T, s)$, is $\max\{\ell(v) \mid v \in V_T\}$. For a graph $G = (V, E)$, and a designated vertex s, the* radius of the graph G with respect to the vertex s, *denoted $Rad(G, s)$, is the radius of any breadth-first-search (henceforth, BFS) spanning tree of the graph G rooted in the vertex s. The* diameter *of a graph $G = (V, E)$, denoted $Diam(G)$, is the maximal distance between a pair of vertices $u, w \in V$. For a subset $C \subseteq V$ of the vertices, the* diameter *of C, denoted $Diam_G(C)$, is the maximum distance in $G(C)$ between a pair of vertices in C, where $G(C) = (C, E(C))$, $E(C) = \{(u, w) \in E \mid u, w \in C\}$ is the* subgraph *of G* induced *by the subset C.*

Lemma 2.5 *For an n-vertex graph rooted tree (T, s) and a vertex $v \in V_T$, $0 \leq rank(v) \leq \log n$.*

Proof: Follows directly from the definition. ▮

Definition 2.6 *For a graph $G = (V, E)$ and a pair of positive integer numbers κ and α, a (κ, α)-*partition *$\mathcal{C}$ of the graph G is a collection of subsets of V, called* clusters, *that satisfy*

1. *For every cluster $C \in \mathcal{C}$, $Diam(C) \leq \kappa$.*
2. *Consider the super-graph $\tilde{G} = (\tilde{V}, \tilde{E})$, $\tilde{V} = \mathcal{C}$, $\tilde{E} = \{(C, C') \mid \exists v \in C,\ v' \in C' \text{ s.t. } (v, v') \in E\}$. The super-graph $\tilde{G}$ is α-*colorable, *i.e., there exists a coloring $\chi : \mathcal{C} \to [\alpha]$ such that for every super-edge $(C, C') \in \tilde{E}$, necessarily $\chi(C) \neq \chi(C')$.*

Theorem 2.7 *[18, 2] For every n-vertex graph $G = (V, E)$, there exists an $(O(\log n), O(\log n))$-partition $\mathcal{C}$ of the graph G. Furthermore, this partition is computable deterministically within time* *polynomial in n, and also, the coloring χ that is associated with the partition $\mathcal{C}$ and satisfies the property 2 of Definition 2.6 is also computable within the same time bounds.*

Definition 2.8 *For an n-vertex rooted tree $(T = (V_T, E_T), s)$, a* semi-spanning path-forest *F of T is a collection of paths that satisfy*

1. *Every path $P \in F$, $P = (V_P, E_P)$, is contained in the tree T, i.e., $V_P \subseteq V_T$, $E_P \subseteq E_T$, and, furthermore, if $V_P = (v_1, v_2, \ldots, v_k)$ then either v_1 is an ancestor of v_k in the tree T or vice versa.*
2. *The paths of the forest F are vertex-disjoint.*
3. *For every vertex $v \in V_T$, the unique path that connects the vertices s and v in the tree T contains at most $\log n$ edges that do not belong to the set $\bigcup_{P \in F} E_P$ (such edges will be henceforth referred to as the* non-edges of the forest *F).*

Lemma 2.9 *For every n-vertex rooted tree (T, s), a semi-spanning path-forest F of the tree exists, and can be constructed within time polynomial in n.*

Proof: Consider the edge set $E' = \{(v, w) \in E_T \mid rank(v) = rank(w)\}$. Observe that the edge set E' decomposes into a collection of vertex-disjoint paths. Denote this collection by F. It is easy to see that F is a semi-spanning path-forest of the input tree. ∎

Definition 2.10 *For a schedule $\Pi = (R_1, R_2, \ldots, R_t)$, a vertex v is said to be* scheduled to broadcast on round $\ell \in [t]$ by the schedule Π, *if $v \in R_\ell$.*

3 Preprocessing

Our algorithm consists of two major parts, the *preprocessing* algorithm, and the algorithm that actually constructs the schedule. While neither of the algorithms is executed offline, and running them efficiently is equally important, the preprocessing algorithm is somewhat detached from the specifics of the radio broadcast problem, and this motivates its name.

In this section we describe our preprocessing algorithm, which is essentially a variant of the analogous part of the algorithm due to Gaber and Mansour [12]. However, unlike the algorithm of [12] our algorithm forms *overlapping layers* and not disjoint ones, and this difference affects the implementation. In terms of its effect on the length of the resulting schedule, this different way of implementing the preprocessing algorithm enables us to reduce the additive term from $O(\log^5 n)$ in the result of [12] to $O(\log^4 n)$.

The input of the radio broadcast problem is a graph $G = (V, E)$, and a source vertex $s \in V$. Let *Rad* denote *Rad*(G, s). Assume that *Rad* is at least a sufficiently large constant. For graphs with smaller radius optimal up to a constant factor schedules are provided by the algorithms of Chlamtac and Weinstein [6] and Bar-Yehuda et al. [4].

The first step of the preprocessing algorithm is to construct a BFS spanning tree T of the graph G, rooted in the source s. The second step of the algorithm is to cover the vertex set V by x or $x+1$ subsets, called *layers*, where $x = 1, 2, \ldots$ is a positive integer parameter, $x \leq Rad/3$. The specific value of x will be determined later. Let d denote $\lfloor Rad/x \rfloor$. Note that $d \geq 3$. The vertex set V is covered by the subsets $V_1, V_2, \ldots, V_x$, (and, possibly V_{x+1} if Rad is not divisible by x), where $V_j = \{v \mid (j-1)d + 1 \leq \ell(v) \leq jd\}$, and $\ell(v) = d_G(s, v)$ is the *level* of the vertex v. Set $y = x$ if Rad is divisible by x, and $y = x + 1$ otherwise. Note that for every index $j \in [y-1]$, the subsets V_j and V_{j+1} intersect.

The third step of the algorithm is to form a set $S(v)$ of vertices for every vertex v that satisfies $\ell(v) \equiv 0 \pmod d$. The set $S(v)$ contains all the descendents w of the vertex v in the tree T that satisfy $\ell_T(w) \leq \ell_T(v) + d$.

Let $\tilde{V} = \{S(v) \mid \ell(v) \equiv 0 \pmod d\}$, and $\tilde{E} = \{(S(v), S(v')) \mid (S(v) \cap S(v') \neq \emptyset)$ or $(\exists w \in S(v),\ w' \in S(v')$ s.t. $(w, w') \in E$ and $\ell_T(v) = \ell_T(v'))\}$. Let $\tilde{G} = (\tilde{V}, \tilde{E})$. Note that while two subsets $S(v)$, $S(v')$ with $\ell_T(v) = \ell_T(v')$ share a super-edge if $S(v) \cap S(v') \neq \emptyset$, *or* if there exists and edge $(w, w') \in E$, $w \in S(v)$, $w' \in S(v')$, two subsets $S(v)$, $S(v')$ with $\ell(v) \neq \ell(v')$ share a super-edge only if they intersect.

For every index $j \in [y]$, let $\tilde{V}_j$ denote the subset of $\tilde{V}$ that contains only the sets $S(v) \in \tilde{V}$ that satisfy $\ell_T(v) = (j-1)d$. Let $\tilde{E}_j = \tilde{E}(\tilde{V}_j)$ denote the subset of the super-edges that is induced by the subset $\tilde{V}_j$ of the super-vertices. Denote $\tilde{G}_j = (\tilde{V}_j, \tilde{E}_j)$. Next, for every index $j \in [y]$, the preprocessing algorithm forms an $(O(\log n), O(\log n))$-partition $\mathcal{C}_j$ of the super-graph $\tilde{G}_j$. Let $\mathcal{C} = \bigcup_{j=1}^{y} \mathcal{C}_j$. Note that for an index $j \in [y]$, and a cluster $C \in \mathcal{C}_j$, the diameter of the cluster C with respect to the super-graph $\tilde{G}_j$ is $O(\log n)$, and so, the diameter of the cluster $C_G = \bigcup_{S(v) \in C} S(v)$ is $O(d \log n)$ (since $Diam_G(S(v)) \leq 2d$). With a slight abuse of notation we will henceforth denote by C both the collection of subsets $S(v)$ that were used to form the cluster C, and the set $C_G = \bigcup_{S(v) \in C} S(v)$.

Note that every cluster $C \in \mathcal{C}_j$ contains one or more vertex v of level $\ell_T(v) = (j-1)d$, and one or more vertex w of level $l_T(w) = \min\{jd, Rad\}$. Note also that viewing clusters as collections of subsets $S(v)$, different clusters $C, C' \in \mathcal{C}_j$ cannot possibly intersect in a subset $S(v)$, because $\mathcal{C}_j$ is a *partition* of the super-graph $\tilde{G}_j$. However, viewing clusters as collections of vertices of the graph G, since the sets $S(v)$ do intersect, the clusters $C, C' \in \mathcal{C}_j$ may intersect in a subset $S = C \cap C' \subseteq V$. For a pair of clusters $C_j \in \mathcal{C}_j$, $C_{j+1} \in \mathcal{C}_{j+1}$, they may intersect in a subset $S = C_j \cap C_{j+1} \subseteq V$ of vertices of level jd. Since $d \geq 3$, it is easy to see that two clusters $C_j \in \mathcal{C}_j$, $C_{j'} \in \mathcal{C}_{j'}$ with $|j - j'| \geq 2$ cannot possibly intersect.

The partition $\mathcal{C}_j$ will be referred to as the *jth layer* of the super-graph $\tilde{G}$. Note that the first layer $\mathcal{C}_1$ of the super-graph $\tilde{G}$ contains only one single cluster $C(s) = S(s)$.

Next, the preprocessing algorithm assigns *ranks* to all the clusters of the collection $\mathcal{C}$, constructs a BFS spanning tree τ rooted in the cluster $C(s)$ of the graph $\tilde{G}$, and chooses a pair of designated vertices, called the *messenger* and the *representative* for every cluster $C \in \mathcal{C}$. These three tasks are performed simultaneously in the following way.

The algorithm initializes the tree τ to be an empty tree. Then, it assigns rank 0 to every cluster $C \in \mathcal{C}_y$. Next, in every cluster $C \in \mathcal{C}_y$, the algorithm picks an arbitrary leaf $w \in C$ of the tree T and designates it to serve as the *representative* of the cluster C. In addition, in every cluster $C \in \mathcal{C}_y$, the algorithm designates one of the ancestors u of the representative w that has level $\ell_T(u) = (y-1) \cdot d$ to serve as the *messenger* of the cluster C. Then the algorithm initializes the set $\mathcal{U}$ of *uncovered* clusters to be equal to $\mathcal{C}_y$.

After the algorithm finishes with the layer $j = 2, 3, \ldots, y$, it proceeds to layer $j-1$. Specifically, it passes over the clusters $C_{j-1} \in \mathcal{C}_{j-1}$ in an arbitrary order. For every cluster $C_{j-1} \in \mathcal{C}_{j-1}$ it does the following. Let $\mathcal{N} \subseteq \mathcal{U}$ be the subset of clusters C whose intersection with the cluster C_{j-1} contains the messenger u of the cluster C. For each cluster $C \in \mathcal{N}$, the algorithm adds the super-edge (C_{j-1}, C) into the tree τ, and sets C_{j-1} to be the parent of C in the tree τ. Let $\mathcal{R}$ be the set of ranks of all the children of the cluster C_{j-1}. Let $r = \max\{r' \in \mathcal{R}\}$. If there exists exactly one cluster $C \in \mathcal{N}$ with rank r, then the messenger w of the cluster C becomes the *representative* of the cluster C_{j-1}. In this case the algorithm assigns the cluster C_{j-1} rank r. Otherwise, an arbitrary cluster $C \in \mathcal{N}$ is chosen, and its messenger w becomes the representative of the cluster C_{j-1}. In this case the algorithm assigns the cluster C_{j-1} rank $r+1$. Now, the algorithm designates one of the ancestors $u \in C_{j-1}$ of the representative w of the cluster C_{j-1} that satisfies $\ell_T(u) = (j-2) \cdot d$ to serve as the messenger of the cluster C_{j-1}. Finally, the set $\mathcal{N}$ is removed from the set $\mathcal{U}$ (i.e., $\mathcal{U} \leftarrow \mathcal{U} \setminus \mathcal{N}$), and the algorithm enters into another iteration of the loop by picking another cluster of layer $j-1$.

Observe that this procedure indeed builds a tree τ rooted in the cluster $C(s)$, and this tree spans the entire collection $\mathcal{C}$ of clusters. Furthermore, note that τ is a BFS spanning tree of the super-graph $\tilde{G}$. Note also that for every cluster $C \in \mathcal{C}$, the rank of the cluster C in the tree τ is exactly equal to the rank that was assigned by this procedure. Note also that the messenger of the cluster $C(s) = S(s)$ is inevitably the source s.

Let $\tilde{E}_{\mathcal{F}}$ be the subset of super-edges $\tilde{e} = (C, C')$ of the tree τ that satisfy $rank(C) = rank(C')$. Note that the set $\tilde{E}_{\mathcal{F}}$ decomposes into a path-forest $\mathcal{F}$, and, furthermore, $\mathcal{F}$ is a semi-spanning path-forest. In addition, the path-forest $\mathcal{F}$ satisfies one more important property.

Lemma 3.1 *Consider a path $\mathcal{P} \in \mathcal{F}$, $\mathcal{P} = (C_1, C_2, \ldots, C_k)$, where C_1 is the ancestor of C_k in the tree τ. For each index $i \in [k]$, let u_i and w_i denote the messenger and the representative of the cluster C_i, respectively, and let P_i denote the path between them in the tree T. Then the paths P_i can be concatenated to form a single path $P = P_1 \cdot P_2 \cdot \ldots \cdot P_k$ in the tree T, the vertex u_1 is an ancestor of the vertex w_k in the tree T, and $u_i = w_{i-1}$ for every index $i \in [k] \setminus \{1\}$.*

Note that the number of distinct subsets $S(v)$ is at most n, and therefore, $|\mathcal{C}| \leq |\tilde{V}| \leq n$, as well. Note also that for every cluster $C \in \mathcal{C}$, the path between the root cluster $C(s)$ and the cluster C in the tree τ contains at most $\log |\mathcal{C}| \leq \log n$ non-edges of the forest $\mathcal{F}$.

Also, consider two clusters $C, C' \in \mathcal{C}_j$, for some index $j \in [y]$. Let u (resp., u') be the messenger of the cluster C (resp., C'), and w (resp., w') be its representative. Recall that $\mathcal{C}_j$ is an $(O(\log n), O(\log n))$-partition of the graph $\tilde{G}_j = (\tilde{V}_j, \tilde{E}_j)$, and $C, C' \in \tilde{V}_j$. Let χ_j be the coloring associated with this partition, and suppose that $\chi_j(C) = \chi_j(C')$. Note that it implies that $C \cap C' = \emptyset$, and, furthermore, there exists no edge $e = (z, z') \in E$, with $z \in C$, $z' \in C'$. Recall also that $|\{\chi_j(C) \mid C \in \mathcal{C}_j\}| \leq \alpha = O(\log n)$.

4 Constructing the schedule

Our algorithm for constructing the schedule is given as input the description of the layers $\mathcal{C}_1, \mathcal{C}_2, \ldots, \mathcal{C}_y$, the colorings $\chi_1, \chi_2, \ldots, \chi_y$ of the layers, the tree τ, the path-forest $\mathcal{F}$, and the messenger-representative pair of vertices for every cluster. All this is the output of the preprocessing algorithm.

The algorithm for constructing the schedule uses two major building blocks, namely, Procedures *Vertical_Broadcast* and *Horizontal_Broadcast*. These procedures are described in Sections 5 and 6. The two procedures are analogous to Procedures *Broadcast_Through* and *Broadcast_All* of [12]. However, our algorithm does not have an analogue to Procedure *Superlevel_to_Superlevel* of [12], because our preprocessing algorithm forms *overlapping layers*, and not disjoint ones as the algorithm of [12] does. Therefore, in our algorithm the delivery of the message between consequent layers is done via the boundary vertices that participate in both layers.

We remark that while our preprocessing algorithm and the analogous part of the algorithm of [12] have a similar nature, our algorithm for constructing the schedule is substantially different from the analogous part of the algorithm [12], which is called there the "main algorithm". The difference concerns mostly the way that different building blocks are combined into a single schedule. The main algorithm of [12] uses a round-robin technique of time sharing for running different procedures, and this way it ensures that there is no collisions between the executions of different procedures. This technique gives rise to a relatively simple analysis, but it carries a price of increasing the length of the schedule by a constant multiplicative factor. Our algorithm is geared to avoid this unnecessary multiplicative factor in order to achieve a *purely additive* approximation of the optimal schedule. To this end, our algorithm carefully combines different building blocks in such a manner that they cause no collision even though they employ *the same rounds*.

5 Procedure *Vertical_Broadcast*

The simplest setting in which Procedure *Vertical_Broadcast* can be invoked is on a cluster $C \in \mathcal{C}$, assuming that the messenger u of C is informed. In this case the procedure propagates the message along one of the shortest paths in the cluster C between the messenger u and the representative w of C. Observe that this process of propagating the message results in a schedule of length at most d.

The more elaborated, and by far more common, scenario in which Procedure *Vertical_Broadcast* is invoked is as follows. Let $\mathcal{S} = \{C^{(1)}, C^{(2)}, \ldots, C^{(k)}\} \subseteq \mathcal{C}_j$ be a collection of clusters of layer j, for some index $j \in [y]$. Let $u^{(i)}, w^{(i)} \in C^{(i)}$ be the messenger and the representative of the cluster $C^{(i)}$, respectively, for each index $i \in [k]$. Suppose that all the messengers $u^{(1)}, u^{(2)}, \ldots, u^{(k)}$ are informed. Procedure *Vertical_Broadcast*, if it is invoked in this setting, needs to propagate the message to the representatives $w^{(1)}, w^{(2)}, \ldots, w^{(k)}$. For each index $i \in [k]$, the procedure picks some shortest path $P^{(i)}$ between the messenger $u^{(i)}$ and the representative $w^{(i)}$. This path $P^{(i)}$, denoted also as $P(C^{(i)})$, will be henceforth referred to as the *chosen path* of the cluster $C^{(i)}$. Let χ_j be the coloring associated with layer j of the collection $\mathcal{C}$. Let $\mathcal{S}_1, \mathcal{S}_2, \ldots, \mathcal{S}_\alpha$ be the partition of the subset $\mathcal{S}$ into a disjoint union of color classes, that is, $\bigcup_{p=1}^{\alpha} \mathcal{S}_p = \mathcal{S}$, with $\mathcal{S}_p = \{C \in \mathcal{S} \mid \chi_j(C) = p\}$ for each index $p \in [\alpha]$. Note that some color classes $\mathcal{S}_p$ may be empty. Observe that for an index $p \in [\alpha]$, and for a pair of distinct clusters $C, C' \in \mathcal{S}_p$, necessarily $C \cap C' = \emptyset$, and, furthermore, the edge set E of the graph G contains no edge (z, z') with $z \in C$ and $z' \in C'$. In particular, the same property holds for the vertex sets V_P and $V_{P'}$ of the paths P and P' respectively. Recall that the number α of color classes is $O(\log n)$.

For a path $P = (u = u_1, u_2, \ldots, u_{d'} = w)$, $d' \leq d$, between the messenger u and the representative w of some cluster C, and an index $\ell \in [d']$, the vertex u_ℓ is called the *ℓth vertex* of the path P.

We next describe the structure of the schedule that Procedure *Vertical_Broadcast* forms whenever it is invoked in this scenario. On the first round of this schedule, the first vertices of the paths $P(C)$ for different clusters $C \in \mathcal{S}_1$ broadcast. (In a situation like this, when the broadcast of the vertex u_1 was originated as part of processing the cluster C, we say that the vertex u_1 *C-broadcasts*. For a vertex v that belongs to an intersection of two clusters C and C', in principle, it may happen that the vertex v both C- and C'-broadcasts in the same round. This, of course, would have the same effect as if it would be just C-broadcast, or just C'-broadcast, and the distinction between these situations is needed only for the analysis.)

More generally, for a round $\ell \in [d]$, on the ℓth round of the schedule that is constructed by Procedure *Vertical_Broadcast*, the ℓth vertices u_ℓ of the chosen paths $P = P(C)$ broadcast, for every cluster $C \in \mathcal{S}_1$. (If one or more of these paths are not long enough to have the ℓth vertex, the set of broadcasting vertices on round ℓ will not contain a vertex from this specific path; if none of these paths contains ℓth vertex, the round will be replaced by a *void* (that is, empty) round.) In addition, for a round $\ell = 4, 5, \ldots, d+3$, on the ℓth round of the schedule that is constructed by Procedure *Vertical_Broadcast*, the $(\ell-3)$rd vertices $u_{\ell-3}$ of the chosen paths $P(C)$ broadcast, for all the clusters $C \in \mathcal{S}_2$. More generally, for an index $p \in [\alpha]$, a round $\ell = 3(p-1)+1, 3(p-1)+2, \ldots, 3(p-1)+d$, on the ℓth round of this schedule the $(\ell - 3(p-1))$th vertices $u_{\ell-3(p-1)}$ of the chosen paths $P(C)$ broadcast, for all the clusters $C \in \mathcal{S}_p$.

This completes the description of the schedule that is constructed by Procedure *Vertical_Broadcast*. We next turn to its analysis.

Observe that the broadcast in clusters of the same color cannot possibly cause a collision. Consider a pair of clusters $C, C' \in \mathcal{C}_j$ with $\chi_j(C) = p$, $\chi_j(C') = q$, and $p \neq q$. Consider some round $\ell = 1, 2, \ldots, d+3(\alpha-1)$. Let $v \in C$, $v' \in C'$ be a pair of vertices that are scheduled to C-broadcast and C'-broadcast on round ℓ, respectively. It follows that v is the $(\ell-3(p-1))$th vertex of the path $P(C)$, and that v' is the $(\ell-3(q-1))$th vertex of the path $P(C')$. Recall that $C, C' \in \mathcal{C}_j$. It follows that $\ell_T(v) = (j-1)d + \ell - 3(p-1)$, and $\ell_T(v') = (j-1)d + \ell - 3(q-1)$, and so, since $p \neq q$ are both integer, $|\ell_T(v) - \ell_T(v')| \geq 3$. In other words, $|d_G(s,v) - d_G(s,v')| \geq 3$, and so $d_G(v,v') \geq 3$. Hence, the broadcasts of the vertices v and v' do not cause a collision, as required.

Hence, executions of Procedure *Vertical_Broadcast* do not cause collisions between vertices that participate in the same execution. The length of the schedule that is formed by an execution of Procedure *Vertical_Broadcast* is $\tilde{d} = d + 3(\alpha - 1)$. Note that by padding the schedules with as many void rounds as required we guarantee that every execution of Procedure *Vertical_Broadcast* forms a schedule of length precisely $\tilde{d}$. This homogeneity of the lengths of schedules is crucial for our analysis. Note also that our description of Procedure *Vertical_Broadcast* assumes that all the clusters in which the broadcast should be conducted are known in advance, and no cluster ever joins "on the fly". In our analysis we will show that our algorithm invokes Procedure *Vertical_Broadcast* in such a way that this property is indeed satisfied.

We also remark that all the procedures that our algorithm employs accept as input an additional input parameter, called the *starting round*. This parameter is a positive integer number that determines the index of the first non-empty round of the schedule that the procedure should construct. In other words, if according to the above description Procedure *Vertical_Broadcast* is supposed to return a schedule $\Pi = (R_1, R_2, \ldots, R_t)$ on a certain input, then whenever it is invoked on the same input with a starting round parameter t' it would return the schedule $\epsilon_{t'-1} \cdot \Pi$, where $\epsilon_{t'-1}$ stands for a schedule that contains $t'-1$ void rounds, and $\cdot$ stands for concatenation. For an execution χ of one of the procedures that are used by our algorithm, its starting round will be denoted by $I(\chi)$.

6 Procedure *Horizontal_Broadcast*

Consider a collection of clusters $\mathcal{S} = \{C^{(1)}, C^{(2)}, \ldots, C^{(k))}\} \subseteq \mathcal{C}_j$, where $j \in [y]$ is an index. Suppose that all the representatives $w^{(1)}, w^{(2)}, \ldots, w^{(k)}$ of these clusters are informed, and the goal is to inform all the vertices of the set $\bigcup_{C \in \mathcal{S}} C$. This is done using Algorithm *Anonymous_Broadcast* due to Bar-Yehuda et al. [4]. Specifically, we use the following result.

Theorem 6.1 *[4, 16] Given an n-vertex graph $\hat{G} = (\hat{V}, \hat{E})$, and a subset $\hat{U} \subseteq \hat{V}$ of informed vertices, there exists a schedule of length $K \cdot (D \log n + \log^2 n)$ that delivers the message to all the vertices of the graph, where K is a universal constant and $D = \max_{v \in V} \min_{u \in U} d_G(u, v)$. Furthermore, this schedule can be computed in deterministic polynomial time.*

Observe that for a cluster $C \in \mathcal{S}$, its diameter in the super-graph $\tilde{G}$ is $O(\log n)$, and each vertex $S(v)$ from which the super-vertices of $\tilde{G}$ were formed is itself a subset of diameter at most $2d$ (in the graph G). Hence, $D \le \max\{Diam_G(C) \mid C \in \mathcal{S}\} = O(d \log n)$, and so the length of the schedule that is constructed by Algorithm *Anonymous_Broadcast* is $O(d \log^2 n)$. Let c denote the universal constant hidden by the O-notation. For the sake of homogeneity of the lengths of schedules constructed by Procedure *Horizontal_Broadcast*, whenever the resulting schedule is of length $L < cd \log^2 n$, it is padded with $cd \log^2 n - L$ empty rounds. By Theorem 6.1, this schedule informs all the vertices of the set $\bigcup_{C \in \mathcal{S}} C$.

To prevent collisions between simultaneous executions of Procedure *Horizontal_Broadcast* in adjacent layers we modify the procedure in the following way. Whenever the input collection of clusters $\mathcal{S}$ is contained in an *even* (resp., *odd*) layer $\mathcal{C}_j$ (i.e., its index j is even (resp., odd)), the procedure constructs a schedule that employs only even (resp., odd) rounds. The odd (resp., even) rounds are left empty. This raises the length of the schedules constructed by Procedure *Horizontal_Broadcast* to $2c \cdot d \log n$. Let a denote the constant $2c$. (Note that this is an example of using the round-robin time sharing technique. In the algorithm of [12] it is used also for avoiding collisions between executions of different procedures, and between different executions of the same procedure that accept as input collections of clusters of the same layer.)

7 Procedure *Period*

In this section we combine Procedures *Vertical_Broadcast* and *Horizontal_Broadcast* described in the previous sections, and describe Procedure *Period*, which is the major component of our algorithm for constructing the schedule.

This procedure is invoked in the same scenario as Procedure *Vertical_Broadcast*, i.e., when it is given a collection $\mathcal{S} \subseteq \mathcal{C}_j$ of clusters of layer j, for some index $j \in [y]$, whose messengers are all informed. Let I denote its starting round parameter.

Procedure *Period* starts with running Procedure *Vertical_Broadcast* on the collection $\mathcal{S}$ with the same starting round parameter I. Let Π_{VL} denote the resulting schedule. Then it forms $\tilde{d}$ void rounds, and then it runs Procedure *Horizontal_Broadcast* on the same collection $\mathcal{S}$, this time with starting round parameter $I + 2\tilde{d}$. Let Π_{HL} denote the resulting schedule. (Note that at this point all the representatives of the clusters $C \in \mathcal{S}$ are informed.) Finally, it forms $2\tilde{d}$ additional void rounds. In addition, Procedure *Period* invokes itself recursively twice, as will be described below.

Note that Procedure *Period* constructs a schedule of length $4\tilde{d} + a \cdot d \log^2 n$ that delivers the message from the set of the messengers of the clusters $C \in \mathcal{S}$ to all the vertices of the set $\bigcup_{C \in \mathcal{S}} C$.

Let $\mathcal{S}'' \subseteq \mathcal{S}$ be a collection of clusters $C \in \mathcal{C}_j$ that have a child (with respect to the tree τ) C' such that the super-edge (C, C') belongs to the path-forest $\mathcal{F}$. Let $\mathcal{S}' = \{C' \mid parent_\tau(C') \in \mathcal{S}''\}$. Note that $\mathcal{S}' \subseteq \mathcal{C}_{j+1}$. Let W'' be the set of representatives of the clusters from the collection $\mathcal{S}''$. Observe that, by construction, each representative $w \in W''$ of a cluster $C \in \mathcal{S}''$ is the messenger of the cluster $C' \in \mathcal{S}'$ that satisfies $parent_\tau(C') = C$. It follows that after the schedule that is returned by the invocation of Procedure *Vertical_Broadcast* terminates, all the messengers of the clusters $C' \in \mathcal{S}'$ are informed.

Procedure *Period* invokes itself recursively on the collection $\mathcal{S}' \subseteq \mathcal{C}_{j+1}$, and sets the starting round parameter of this recursive invocation to be $I + \tilde{d}$.

Let $\hat{\mathcal{S}}$ be the collection of clusters of layer $j+1$ that do not belong to the collection $\mathcal{S}'$, and whose messengers are informed by the schedule that is constructed by the execution of Procedure *Horizontal_Broadcast* on input $\mathcal{S}$. The last step of Procedure *Period* is to invoke itself recursively on the collection $\hat{\mathcal{S}} \subseteq \mathcal{C}_{j+1}$ with starting round parameter $I + 4\tilde{d} + a \cdot d \log^2 n$.

Note that one of the collections $\hat{\mathcal{S}}$, $\mathcal{S}'$, or both of them, could be empty. This would lead to an invocation of Procedure *Period* on an empty collection; in this case it returns an empty schedule. This is, in fact, the termination condition of the recursion.

After the recursive invocations are over, the original execution of Procedure *Period* has three schedules Π_0, Π_1 and Π_2. The schedule Π_0 is obtained by processing the collection $\mathcal{S}$ directly, that is, invoking on it Procedures *Vertical_Broadcast* and *Horizontal_Broadcast* and concatenating them in the way that was described above (along with introducing the void rounds). The schedules Π_1 and Π_2 are returned by the first and the second recursive invocations of Procedure *Period*, respectively (on collections $\mathcal{S}'$ and $\hat{\mathcal{S}}$, respectively). This three schedules are unified into a single schedule Π that the execution returns in a straightforward way. Specifically, let $L = \max\{|\Pi_0|, |\Pi_1|, |\Pi_2|\}$, and let $L_i = |\Pi_i|$, for $i \in \{0,1,2\}$. For each index $i \in \{0,1,2\}$ such that $L_i < L$, pad the schedule Π_i with $L - L_i$ void rounds. For $i \in \{0,1,2\}$, let Π'_i denote the obtained schedules. Now set Π to be the *round-wise* union of the schedules Π'_0, Π'_1, and Π'_2. I.e., let $\Pi'_i = (R_1^{(i)}, R_2^{(i)}, \ldots, R_L^{(i)})$, $i \in \{0,1,2\}$. Then $\Pi = (\bigcup_{i=0}^2 R_1^{(i)}, \bigcup_{i=0}^2 R_2^{(i)}, \ldots, \bigcup_{i=0}^2 R_L^{(i)})$ is the schedule returned by the execution. (Henceforth we refer to this operation of unifying schedules of not necessarily equal lengths as *schedule-union*. In this terminology, the schedule Π is just the schedule-union of schedules Π_0, Π_1

and Π_2. Also, the schedule Π_0 is the schedule-union of the schedules Π_{VL} and Π_{HL}.)

8 The Combined Algorithm

Consider the execution of Procedure *Period* on the input collection that contains only the single cluster $C(s) \in \mathcal{C}_1$ with starting round parameter equal to 1. We next modify the schedule Π that is obtained by this invocation to construct the resulting schedule Π^* of our algorithm.

Let Φ be the set of all the recursive invocations of Procedure *Period* that were originated directly or indirectly by the execution of Procedure *Period* on the input collection $\{C(s)\}$. Let E_Φ be the set of pairs of executions $(\varphi, \varphi') \in \Phi^2$, $\varphi \neq \varphi'$, such that either the execution φ recursively invokes the execution φ' or vice versa. Observe that the graph $T_\Phi = (\Phi, E_\Phi)$ is a binary tree. Consider an internal vertex (that is, not a leaf) $\varphi \in \Phi$ of this tree. This vertex corresponds to an execution of Procedure *Period* on an input collection $\mathcal{S} \subseteq \mathcal{C}_j$ of clusters, for some index $j \in [y-1]$. Since φ is not a leaf of the tree T_Φ, it follows that φ has two children in T_Φ. These children correspond to the executions φ' and $\hat{\varphi}$ of Procedure *Period* on input collections $\mathcal{S}'$ and $\hat{\mathcal{S}}$, respectively. The execution φ' (resp., $\hat{\varphi}$) will be referred as the *first* (resp., *second*) *child-invocation of the execution* φ.

For a vertex v that belongs to a cluster C of layer 1 or 2, its behavior in the schedule Π^* is identical to its behavior in the schedule Π. For a layer $j = 3, 4, \ldots, y$, let $\{\varphi^{(1)}, \varphi^{(2)}, \ldots, \varphi^{(k)}\} = \Phi(j)$ be the set of all different recursive invocations of Procedure *Period* on collections $\mathcal{S} \subseteq \mathcal{C}_j$ of clusters of layer j. In other words, $\Phi(j)$ is the set of vertices φ of the tree T_Φ that have level j, i.e., satisfy $\ell_{T_\Phi}(\varphi) = j$.

Recall that each such execution φ has an attached starting round parameter. For a positive integer $\ell = 1, 2, \ldots$, let Φ_ℓ be the subset of the set Φ that contains all the invocations with starting rounds equal to ℓ. For each non-empty subset Φ_ℓ, let $\mathcal{S}'_\ell = \bigcup\{\mathcal{S} \mid \mathcal{S}$ is the input collection of $\varphi \in \Phi_\ell\}$. Let φ'_ℓ be the execution of Procedure *Period* on the input collection $\mathcal{S}'_\ell$ with starting round parameter equal to ℓ. Now the algorithm replaces the partial schedules that were obtained by running the executions $\varphi^{(1)}, \varphi^{(2)}, \ldots, \varphi^{(k)}$ by the partial schedules obtained by running the executions $\varphi'_1, \varphi'_2, \ldots$ (only those of them for which $\mathcal{S}'_\ell \neq \emptyset$).

The last step of the algorithm is to find the smallest index t such that the first t rounds of the schedule inform all the vertices of the graph, and to remove all the rounds with index greater than t from the schedule. This completes the description of our algorithm.

The following lemma shows that the executions φ'_ℓ are valid executions of Procedure *Period*.

Lemma 8.1 *All the executions φ'_ℓ are within the framework of the prescribed scenario for executing Procedure Period.*

Proof: Observe that for every execution $\varphi \in \Phi_\ell$, its starting round is ℓ, and its input collection is

$\mathcal{S} \subseteq \mathcal{C}_j$, where j is the index such that $\varphi \in \Phi(j)$. Since the execution φ is within the framework of the prescribed scenario for executing Procedure *Period*, it follows that all the messengers of the cluster $C \in \mathcal{S}$ are informed before the starting round ℓ of the execution φ.

Hence, all the messengers of the clusters $C \in \mathcal{S}'_\ell$ are informed before the beginning of the starting round ℓ of the execution φ'_ℓ. ∎

Lemma 8.2 *The running time of the algorithm is polynomial in n.*

Proof: Consider the child-invocations φ' and $\hat{\varphi}$ of an execution φ of Procedure *Period* on a collection $\mathcal{S} \subseteq \mathcal{C}_j$, for some index $j \in [y-1]$. Consider the input collections $\mathcal{S}', \hat{\mathcal{S}} \subseteq \mathcal{C}_{j+1}$ of the executions φ' and $\hat{\varphi}$, respectively, of Procedure *Period*. Note that by construction, $\mathcal{S}' \cap \hat{\mathcal{S}} = \emptyset$.

A straightforward induction on the level of the specific cluster C in the tree τ shows that every cluster $C \in \mathcal{C} = \bigcup_{j=1}^{y} \mathcal{C}_j$, there is at most one execution φ of Procedure *Period* whose input collection $\mathcal{S}$ contains the cluster C. Therefore, the overall number of (recursive) invocations of Procedure *Period* throughout the execution *Period*$(\{C(s)\})$ is at most $|\mathcal{C}|$.

Since every cluster C of $\mathcal{C}$ contains at least one vertex that belongs to no other cluster $C' \in \mathcal{C}$ (recall that $d \geq 3$), it follows that $|\mathcal{C}| \leq n$, and so at most n recursive invocations of Procedure *Period* occur during the execution of the algorithm. Since every invocation of Procedure *Period* (disregarding its child-invocations) requires running time at most polynomial in n, it follows that the running time of the entire algorithm is polynomial in n as well. (Notice also that the step of our algorithm that merges some of the executions of Procedure *Period* can also be carried through in polynomial time.) ∎

9 Analysis

In this section we argue that our algorithm constructs a short broadcast schedule that informs all the vertices of the input graph G.

We start with the following lemma that is proven by induction on the indices $j \in [y]$ of layers $\mathcal{C}_j$ of the super-graph $\tilde{G}$.

Lemma 9.1 *The starting rounds of executions φ and γ of Procedures Period and Vertical_Broadcast, respectively, on input collections $\mathcal{S} \subseteq \mathcal{C}_j$ are of the form*

$$I(\varphi) \;=\; I(\gamma) \;=\; 1 + \tilde{d} \cdot (j-1) + h(3 \cdot \tilde{d} + a \cdot d \log^2 n) \;,$$

for some positive integer $h \in [j-1]$. The starting rounds of executions δ of Procedure Horizontal_Broadcast on such a collection are of the form

$$I(\delta) \;=\; 1 + \tilde{d} \cdot (j+1) + h(3 \cdot \tilde{d} + a \cdot d \log^2 n) \;,$$

for some positive integer $h \in [j-1]$.

Proof:

The induction base: For layer $j = 1$, the only execution of Procedure *Vertical_Broadcast* (on the input collection $\{C(s)\}$) has starting round 1, and the same is true for Procedure *Period*. The only execution of Procedure *Horizontal_Broadcast* (on the same singleton input collection) starts at round $1 + 2\tilde{d}$, proving the induction base.

The induction step: For an index $j \in [y]$, let $Period_j$ (resp., $Vertical\_Broadcast_j$; $Horizontal\_Broadcast_j$) denote the set of all the executions of Procedure *Period* (resp., *Vertical_Broadcast*; *Horizontal_Broadcast*) on different collections $\mathcal{S} \subseteq \mathcal{C}_j$ that occur during the algorithm.

For an execution $\varphi \in Period_j$, let $\gamma(\varphi)$ (resp., $\delta(\varphi)$) denote the execution of Procedure *Vertical_Broadcast* (resp., *Horizontal_Broadcast*) that is invoked by φ. Let φ' and $\hat{\varphi}$ denote the first and second child-invocations of the execution φ (note that $\varphi', \hat{\varphi} \in Period_{j+1}$).

By the induction hypothesis, since $\varphi \in Period_j$,

$$I(\varphi) = 1 + \tilde{d} \cdot (j-1) + h(\varphi) \cdot (3 \cdot \tilde{d} + a \cdot d \log^2 n) ,$$

where $h(\varphi) \in [j-1]$. Also, by construction,

$$\begin{aligned} I(\varphi') &= I(\varphi) + \tilde{d} \ , \ I(\hat{\varphi}) \ = \ I(\varphi) + 4\tilde{d} + a \cdot d \log^2 n \ , \\ I(\gamma(\varphi)) &= I(\varphi) \ , \ I(\delta(\varphi)) \ = \ I(\varphi) + 2 \cdot \tilde{d} \ , \\ I(\gamma(\varphi')) &= I(\varphi') \ , \ I(\delta(\varphi')) \ = \ I(\varphi') + 2 \cdot \tilde{d} \ , \\ I(\gamma(\hat{\varphi})) &= I(\hat{\varphi}) \ , \ I(\delta(\hat{\varphi})) \ = \ I(\hat{\varphi}) + 2 \cdot \tilde{d} \ . \end{aligned}$$

Observe that $\varphi \in Period_j$, $\varphi', \hat{\varphi} \in Period_{j+1}$, $\gamma(\varphi'), \gamma(\hat{\varphi}) \in Vertical\_Broadcast_{j+1}$, and $\delta(\varphi'), \delta(\hat{\varphi}) \in Horizontal\_Broadcast_{j+1}$. Hence,

$$\begin{aligned} I(\gamma(\varphi')) &= I(\varphi') \ = \ I(\varphi) + \tilde{d} \\ &= 1 + \tilde{d} \cdot j + h(\varphi) \cdot (3\tilde{d} + a \cdot d \log^2 n) \ , \end{aligned}$$

with $h(\varphi) \in [j-1] \subseteq [j]$.

Analogously,

$$I(\gamma(\hat{\varphi})) \ = \ I(\hat{\varphi}) \ = \ I(\varphi) + 4\tilde{d} + a \cdot d \log^2 n \ = \ 1 + \tilde{d} \cdot j + h(\gamma(\hat{\varphi})) \cdot (3\tilde{d} + a \cdot d \log^2 n) \ ,$$

with $h(\gamma(\hat{\varphi})) = h(\varphi) + 1$. Since $h(\varphi) \in [j-1]$, it follows that $h(\gamma(\hat{\varphi})) \in [j]$.

Also,

$$\begin{aligned} I(\delta(\varphi')) &= I(\varphi') + 2\tilde{d} \ = \ I(\varphi) + 3\tilde{d} \\ &= 1 + \tilde{d} \cdot (j+2) + h(\varphi) \cdot (3\tilde{d} + a \cdot d \log^2 n) \ , \end{aligned}$$

and $h(\varphi) = h(\delta(\varphi)) \in [j-1] \subseteq [j]$.

Finally,

$$\begin{aligned} I(\delta(\hat{\varphi})) &= I(\hat{\varphi}) + 2\tilde{d} = I(\varphi) + 6\tilde{d} + a \cdot d \log^2 n \\ &= 1 + \tilde{d} \cdot (j+2) + (h(\varphi)+1) \cdot (3\tilde{d} + a \cdot d \log^2 n) \ , \end{aligned}$$

and so $h(\delta(\hat{\varphi})) = h(\varphi) + 1 \in [j]$, as required.

To summarize, $I(\gamma(\varphi'))$, $I(\gamma(\hat{\varphi}))$, $I(\delta(\varphi'))$ and $I(\delta(\hat{\varphi}))$ are all of required form. (Note that $\gamma(\varphi'), \gamma(\hat{\varphi}) \in \mathit{Vertical\_Broadcast}_{j+1}$, and $\delta(\varphi'), \delta(\hat{\varphi}) \in \mathit{Horizontal\_Broadcast}_{j+1}$, and the required forms for the staring rounds of executions of Procedures *Vertical_Broadcast* and *Horizontal_Broadcast* are different.) ∎

Notation 9.2 *For an execution χ of one of the procedures that are used by our algorithm, let $\Pi(\chi)$ denote the schedule returned by this procedure.*

Definition 9.3 *Consider two executions χ_1, χ_2 of procedures that are used in our algorithm. These two executions are said to* interfere one with another *if there exists an index $\ell = 1, 2, \ldots$, and a pair of (not necessarily distinct) vertices $u, w \in V$ at distance at most 2 one from another in the graph $G = (V, E)$, such that the vertex u is scheduled to broadcast on round ℓ by the schedule $\Pi(\chi_1)$ and the vertex w is scheduled to broadcast on round ℓ by the schedule $\Pi(\chi_2)$.*

Lemma 9.4 *Different executions of Procedures Vertical_Broadcast and Horizontal_Broadcast do not interfere one to another.*

Proof: Consider two executions χ_1 and χ_2. For $i \in \{1, 2\}$, let j_i be the *layer* of the execution χ_i, that is, the index that satisfies $\chi_i \in \mathit{Vertical\_Broadcast}_{j_i} \cup \mathit{Horizontal\_Broadcast}_{j_i}$. Observe that if $|j_1 - j_2| \geq 2$ then the executions χ_1 and χ_2 cannot interfere one to another. (Recall that $d \geq 3$.) Hence, from now on we assume that either the layers of the executions χ_1 and χ_2, denoted $\ell(\chi_1)$ and $\ell(\chi_2)$, are equal, or they differ by 1.

The discussion splits into two cases. The first case is $\ell(\chi_1) = \ell(\chi_2) = j$, for some index $j \in [y]$, and the second case is $\ell(\chi_1) = j$, $\ell(\chi_2) = j+1$ (without loss of generality), for some index $j \in [y-1]$.

1. $(\ell(\chi_1) = \ell(\chi_2) = j)$
   The discussion splits again, this time into three cases, depending on whether χ_1, χ_2 are both executions of Procedure *Vertical_Broadcast*, or both of them are executions of Procedure *Horizontal_Broadcast*, or one of them is an execution of Procedure *Vertical_Broadcast* and the other is an execution of Procedure *Horizontal_Broadcast*.

   (a) $(\chi_1, \chi_2 \in \mathit{Vertical\_Broadcast}_j)$
   In this case the starting rounds $I(\chi_1)$, $I(\chi_2)$ of the two executions satisfy

   $$I(\chi_1) = 1 + \tilde{d} \cdot (j-1) + h(\chi_1) \cdot (3\tilde{d} + a \cdot d \log^2 n) \ ,$$

$$I(\chi_2) \quad = \quad 1 + \tilde{d} \cdot (j-1) + h(\chi_2) \cdot (3\tilde{d} + a \cdot d \log^2 n) \ ,$$

where $h(\chi_1), h(\chi_2) \in [j-1]$.

If $h(\chi_1) = h(\chi_2)$ then $I(\chi_1) = I(\chi_2)$, but by construction, this cannot happen to two distinct executions of the same procedure that have input collections of the same layer. (Recall that in this case the input collections $\mathcal{S}(\chi_1), \mathcal{S}(\chi_2) \in \mathcal{C}_j$ of these two executions would be merged into a single collection, and Procedure *Vertical_Broadcast* would be invoked on a unified collection that would contain the set $\mathcal{S}(\chi_1) \cup \mathcal{S}(\chi_2)$.)

If $h(\chi_1) \neq h(\chi_2)$ then since both $h(\chi_1)$ and $h(\chi_2)$ are integer, it follows that $|I(\chi_1) - I(\chi_2)| \geq 3\tilde{d} + a \cdot d \log^2 n$. Suppose, without loss of generality, that $I(\chi_1) < I(\chi_2)$. The length of the schedule $\Pi(\chi_1)$ that is constructed by the execution χ_1 is $\tilde{d} < I(\chi_2) - I(\chi_1)$, and so, the executions χ_1 and χ_2 do not interfere one to another.

(b) $(\chi_1, \chi_2 \in \textit{Horizontal\_Broadcast}_j)$

In this case the starting rounds of the two executions satisfy

$$\begin{aligned} I(\chi_1) &= 1 + \tilde{d} \cdot (j+1) + h(\chi_1) \cdot (3\tilde{d} + a \cdot d \log^2 n) \ , \\ I(\chi_2) &= 1 + \tilde{d} \cdot (j+1) + h(\chi_2) \cdot (3\tilde{d} + a \cdot d \log^2 n) \ , \end{aligned}$$

where $h(\chi_1), h(\chi_2) \in [j-1]$.

If $h(\chi_1) = h(\chi_2)$ then $I(\chi_1) = I(\chi_2)$, and by the same argument as in Case 1a, this implies that $\chi_1 = \chi_2$, and we are done.

If $h(\chi_1) \neq h(\chi_2)$ then $|I(\chi_1) - I(\chi_2)| \geq 3\tilde{d} + a \cdot d \log^2 n$. Since $\max\{|\Pi(\chi_1)|, |\Pi(\chi_2)|\} = a \cdot d \log^2 n < |I(\chi_1) - I(\chi_2)|$, the executions χ_1 and χ_2 would not interfere one to another in this case as well.

(c) $(\chi_1 \in \textit{Vertical\_Broadcast}_j,\ \chi_2 \in \textit{Horizontal\_Broadcast}_j)$

In this case

$$\begin{aligned} I(\chi_1) &= 1 + \tilde{d} \cdot (j-1) + h(\chi_1) \cdot (3\tilde{d} + a \cdot d \log^2 n) \ , \\ I(\chi_2) &= 1 + \tilde{d} \cdot (j+1) + h(\chi_2) \cdot (3\tilde{d} + a \cdot d \log^2 n) \ , \end{aligned}$$

where $h(\chi_1), h(\chi_2) \in [j-1]$.

If $h(\chi_1) = h(\chi_2)$ then $I(\chi_2) - I(\chi_1) = 2\tilde{d}$. The length of the schedule $\Pi(\chi_1)$ that is constructed by the execution χ_1 is $\tilde{d} < I(\chi_2) - I(\chi_1)$, and so, in this subcase we are done.

If $h(\chi_1) < h(\chi_2)$ then $I(\chi_1) < I(\chi_2)$ too, and the difference $I(\chi_2) - I(\chi_1)$ is even greater than in the subcase $h(\chi_1) = h(\chi_2)$.

Finally, consider the subcase $h(\chi_1) > h(\chi_2)$. Since $h(\chi_1)$ and $h(\chi_2)$ are both integer, it follows that

$$\begin{aligned} I(\chi_1) - I(\chi_2) &= (h(\chi_1) - h(\chi_2)) \cdot (3\tilde{d} + a \cdot d \log^2 n) - 2\tilde{d} \\ &\geq \tilde{d} + a \cdot d \log^2 n \ > \ 0 \ , \end{aligned}$$

and so, the starting round of the execution χ_2 is at least by $\tilde{d} + a \cdot d \log^2 n$ smaller than the starting round of the execution χ_1. The length of the schedule $\Pi(\chi_2)$ is $a \cdot d \log^2 n$, and so, this schedule terminates $\tilde{d}$ rounds before the starting round of the execution χ_1, implying that the two executions do not interfere one to another.

2. $(\ell(\chi_1) = j,\ \ell(\chi_2) = j + 1)$

   (a) ($\chi_1 \in$ *Vertical_Broadcast*$_j$, $\chi_2 \in$ *Vertical_Broadcast*$_{j+1}$)
   In this case

$$
\begin{aligned}
I(\chi_1) &= 1 + \tilde{d} \cdot (j-1) + h(\chi_1) \cdot (3\tilde{d} + a \cdot d \log^2 n)\ , \\
I(\chi_2) &= 1 + \tilde{d} \cdot j + h(\chi_2) \cdot (3\tilde{d} + a \cdot d \log^2 n)\ ,
\end{aligned}
$$

   where $h(\chi_1) \in [j-1]$, $h(\chi_2) \in [j]$.
   If $h(\chi_1) = h(\chi_2)$ then $I(\chi_2) - I(\chi_1) = \tilde{d}$, which is precisely the length of the schedule $\Pi(\chi_1)$, and so, the schedule $\Pi(\chi_1)$ terminates exactly one round before the starting round of the execution χ_2.
   If $h(\chi_1) \neq h(\chi_2)$ then

$$
|I(\chi_2) - I(\chi_1)| \ \geq\ 2\tilde{d} + a \cdot d \log^2 n \ \geq\ \max\{|\Pi(\chi_1)|, |\Pi(\chi_2)|\} = \tilde{d}\ .
$$

   and, therefore, the two executions do not interfere one to another.

   (b) ($\chi_1 \in$ *Vertical_Broadcast*$_j$, $\chi_2 \in$ *Horizontal_Broadcast*$_{j+1}$)
   In this case

$$
\begin{aligned}
I(\chi_1) &= 1 + \tilde{d} \cdot (j-1) + h(\chi_1) \cdot (3\tilde{d} + a \cdot d \log^2 n)\ , \\
I(\chi_2) &= 1 + \tilde{d} \cdot (j+2) + h(\chi_2) \cdot (3\tilde{d} + a \cdot d \log^2 n)\ ,
\end{aligned}
$$

   where $h(\chi_1) \in [j-1]$, $h(\chi_2) \in [j]$.
   If $h(\chi_1) = h(\chi_2)$ then $I(\chi_2) - I(\chi_1) = 3\tilde{d} > 0$, and since $|\Pi(\chi_1)| = \tilde{d}$, and $I(\chi_1) < I(\chi_2)$, we are done.
   If $h(\chi_1) \neq h(\chi_2)$ then $|I(\chi_2) - I(\chi_1)| \geq a \cdot d \log^2 n$, and since $\max\{|\Pi(\chi_1)|, |\Pi(\chi_2)|\} = a \cdot d \log^2 n$, the schedule with smaller starting round will terminate at least one round before the beginning of the schedule with the larger starting round.

   (c) ($\chi_1 \in$ *Vertical_Broadcast*$_{j+1}$, $\chi_2 \in$ *Horizontal_Broadcast*$_j$)
   In this case

$$
\begin{aligned}
I(\chi_1) &= 1 + \tilde{d} \cdot j + h(\chi_1) \cdot (3\tilde{d} + a \cdot d \log^2 n)\ , \\
I(\chi_2) &= 1 + \tilde{d} \cdot (j+1) + h(\chi_2) \cdot (3\tilde{d} + a \cdot d \log^2 n)\ ,
\end{aligned}
$$

   where $h(\chi_1) \in [j]$, $h(\chi_2) \in [j-1]$.

If $h(\chi_1) = h(\chi_2)$ then $I(\chi_2) = I(\chi_1) + \tilde{d}$, and, in particular, $I(\chi_1) < I(\chi_2)$. Since $|\Pi(\chi_1)| = \tilde{d}$, the schedule $\Pi(\chi_1)$ terminates at least one round before $I(\chi_2)$, that is, the round when the schedule $\Pi(\chi_2)$ starts.

If $h(\chi_1) \neq h(\chi_2)$ then $|I(\chi_1) - I(\chi_2)| \geq 2\tilde{d} + a \cdot d \log^2 n \geq \max\{|\Pi(\chi_1)|, |\Pi(\chi_2)|\}$, and we are done.

(d) ($\chi_1 \in$ *Horizontal_Broadcast*$_j$, $\chi_2 \in$ *Horizontal_Broadcast*$_{j+1}$)
Recall that the schedule that is constructed by an execution $\delta \in$ *Horizontal_Broadcast*$_j$ employs only rounds ℓ that satisfy $\ell \equiv j \pmod 2$. Hence the execution χ_1 employs only rounds with even indices and the execution χ_2 employs only rounds with odd indices, or vice versa. Hence, the two executions do not interfere one to another.

▮

Lemma 9.5 *The length of the schedule Π^* that is constructed by our algorithm is at most Rad + $O(d \cdot \log^3 n + x \cdot \log n)$, and it informs all the vertices of the graph.*

Proof: Consider a vertex $v \in V$. Let $C \in \mathcal{C}$ be one of the clusters that contain the vertex v. Let $\mathcal{P}$ be the (unique) path between the root cluster $C(s)$ and the cluster C in the tree τ. Observe that $\mathcal{P}$ contains at most y super-vertices (clusters), and, therefore, at most x super-edges. Recall that since $\mathcal{F}$ is a *semi-spanning* path-forest, the path $\mathcal{P}$ contains at most $\log n$ super-edges that do not belong to $\mathcal{F}$ (henceforth termed as *non-edges* of $\mathcal{F}$).

Consider the unique representation of the path $\mathcal{P}$ as the concatenation $\mathcal{P} = \mathcal{P}_1 \cdot \tilde{e}_1 \cdot \mathcal{P}_2 \cdot \tilde{e}_2 \cdot \ldots \cdot \mathcal{P}_r \cdot \tilde{e}_r \cdot \mathcal{P}_{r+1}$, where $r \in [\log n]$, and for each index $i \in [r]$, $\mathcal{P}_i$ is a (possibly empty) path path that is contained in one of the paths of the forest $\mathcal{F}$, and the super-edges $\tilde{e}_1, \tilde{e}_2, \ldots, \tilde{e}_r$ are non-edges of the path-forest $\mathcal{F}$. Also, the cluster $C(s)$ is one of the endpoints of the path $\mathcal{P}_1$, if $\mathcal{P}_1 \neq \emptyset$, and if $\mathcal{P}_1 = \emptyset$ then $C(s)$ is one of the endpoints of the super-edge $\tilde{e}_1$. Analogously, the cluster C is one of the endpoints of the path $\mathcal{P}_{r+1}$ if the latter is not empty, and if it is empty that the cluster C is one of the endpoints of the super-edge $\tilde{e}_r$.

For each index $i \in [r+1]$, let $p_i = |\mathcal{P}_i|$. Let $\mathcal{P}_i = (C_1^{(i)}, C_2^{(i)}, \ldots, C_{p_i}^{(i)})$, and for each index $j \in [p_i]$, let u_j, w_j denote the messenger and the representative of the cluster $C_j^{(i)}$, respectively. Observe that by Lemma 3.1, $w_1 = u_2$, $w_w = u_3$, $\ldots, w_{p_i - 1} = u_{p_i}$, and the vertex w_{p_i} is a descendent of the vertex u_1 with respect to the BFS spanning tree T of the graph G.

Consider an invocation of Procedure *Period* on some collection $\mathcal{S}$ that contains the cluster $C_1^{(i)}$. Suppose that this invocation has starting round parameter I. By construction, the procedure invokes itself recursively with starting round parameter $I + \tilde{d}$ rounds on a collection $\mathcal{S}'$ of clusters that contains the cluster $C_2^{(i)}$. The latter recursive invocation again invokes itself recursively with starting round parameter $I + 2\tilde{d}$ on a collection $\mathcal{S}''$ of clusters that contains the cluster $C_3^{(i)}$, etc. To summarize, this line of recursive invocations constructs a partial schedule of length $p_i \cdot \tilde{d}$ that delivers the message from the messenger u_1 of the cluster $C_1^{(i)}$ to the messenger u_{p_i} of the cluster

$C^{(i)}_{p_i}$.

For each non-edge $\tilde{e}_i = (C', C'')$, $i \in [r]$, of the path-forest $\mathcal{F}$, the message is delivered from the messenger of the cluster C' to all the other vertices of this cluster (including the messenger of the cluster C'' that belongs to the intersection $C' \cap C''$) via a schedule that is constructed by an execution of Procedure *Period* on a collection $\mathcal{S}$ of clusters that contains the cluster C'.

Procedure *Period* starts with invoking Procedure *Vertical_Broadcast* on the same input collection $\mathcal{S}$. This invocation constructs a schedule of length $\tilde{d}$ that delivers the message from the messenger u' of the cluster C' to its representative w'. Then this schedule is padded with $\tilde{d}$ empty rounds, and then Procedure *Period* concatenates the resulting schedule with the schedule that is returned by the invocation of Procedure *Horizontal_Broadcast* on the same input collection $\mathcal{S}$ (but with starting round parameter greater by $2\tilde{d}$ than the one that Procedure *Period* accepted as a part of its input). The latter schedule is of length $a \cdot d \log^2 n$, and, in particular, it delivers the message from the representative w' to all the other vertices of the cluster C' including the messenger u'' of the cluster C''. Finally, this schedule is padded with extra $2\tilde{d}$ empty rounds.

To summarize, for each non-edge $\tilde{e}_i = (C', C'')$, $i \in [r]$, of the path-forest $\mathcal{F}$, Procedure *Period* constructs a schedule of length $4\tilde{d} + a \cdot d \log^2 n$ that delivers the message from the messenger u' of the cluster C' to the messenger u'' of the cluster C''.

Altogether, it follows that our algorithm constructs a schedule that delivers the message from the messenger s of the cluster $C(s)$ to the messenger u of the cluster C, and this schedule is of length

$$\sum_{i=1}^{r+1} p_i \tilde{d} + r \cdot (4\tilde{d} + a \cdot d \log^2 n) \;=\; \tilde{d} \sum_{i=1}^{r+1} p_i + r(4\tilde{d} + a \cdot d \log^2 n)) \ .$$

Observe that $|\mathcal{P}| = \sum_{i=1}^{r+1} p_i + r$. Hence, the length of that schedule is equal to $\tilde{d}|\mathcal{P}| + r \cdot (3\tilde{d} + a \cdot d \log^2 n)$. Finally, the delivery of the message from the messenger u of the cluster C to the vertex v (and to all the other vertices of this cluster C) is done via a schedule of length $4\tilde{d} + a \cdot d \log^2 n$ that is constructed by an invocation of Procedure *Period* on a collection of clusters that contains the cluster C.

Hence, altogether, the schedule that is constructed by our algorithm delivers the message from the source s to the vertex v no later than on round $\tilde{d} \cdot (|\mathcal{P}| + 1) + (r+1) \cdot (3\tilde{d} + a \cdot d \log^2 n)$. Observe that the tree τ is of length x, $r \le \log n$, $\alpha = O(\log n)$, and $\tilde{d} = d + O(\log n)$. Hence, the length of the schedule is at most

$$\begin{aligned} &(d + O(\log n))(x+1) + O(\log n)(O(d) + O(\log n) + O(d \log^2 n)) \;= \\ &d \cdot x + O(x \cdot \log n + d \cdot \log^3 n) \;\le\; Rad + O(x \cdot \log n + d \cdot \log^3 n) \ . \end{aligned}$$

Recall that $d = \lfloor Rad/x \rfloor$, and the lemma follows. ∎

Corollary 9.6 *For an input $(G = (V, E), s)$, $s \in V$, with radius $Rad \ge \log^2 n$, running our* *algorithm with input parameter $x = \sqrt{Rad} \cdot \log n$ results in a radio broadcast schedule of length $Rad + O(\sqrt{Rad} \cdot \log^2 n)$. If $Rad < \log^2 n$ setting $x = Rad/3$ ensures a schedule of length $O(Rad \cdot \log n + \log^3 n) = O(\log^3 n)$.*

Note that for graphs of radius $Rad = \Omega(\log^4 n)$, the algorithm provides a radio broadcast schedule of length $O(Rad)$. Observe also that in the range $Rad < \log^2 n$ the result of [4] provides a better upper bound of $O(Rad \log n + \log^2 n) = O(\log^3 n)$.

10 An improved bound for graphs with small genus

In this section we describe a simplified algorithm and its analysis that provide an improved schedules for graphs with small genus.

The algorithm starts with forming the layers $V_1, \ldots, V_x$ (and possibly V_{x+1}) exactly as in Section 3. Then the algorithm forms *disjoint* supersets in the following way. (Note that in the algorithm for general graphs the supersets are not necessarily disjoint.) The only superset on layer 1 is the d-neighborhood of the vertex s (d is set as in Section 3). On layer 2, the supersets are formed by the following procedure. For $i = 1, 2, \ldots$, let V'_i be the set of all vertices at distance exactly i from s in G. Initialize $U = \bigcup_{i=d}^{2d} V'_i$. Examining the vertices of V'_d one by one, for each vertex $v \in V'_d$ the algorithm forms a superset by taking into it the intersection of the d-neighborhood of v with U, and removes all the vertices of this superset from U. Note that some of these supersets may be empty. Discard the empty supersets. Do the same for layers $3, 4, 5, \ldots$.

This way the algorithm constructs a disjoint partition of every layer into supersets of (strong) diameter at most $2d$. As in the algorithm of Section 3, the supersets in adjacent layers may intersect. However, in this version of the algorithm their intersections may contain at most one vertex. For $i = 1, 2, \ldots, x$, let $\tilde{G}_i$ denote the graph induced by the supersets of layer i. That is, the set of vertices of $\tilde{G}_i$ is the set of supersets of layer i, and there is an edge between two supersets C and C' if and only if there is an edge between some vertex $v \in C$ and $v' \in C'$ in the original graph G. Let χ_i denote the chromatic number of $\tilde{G}_i$, and let $\chi = \max\{\chi_i \mid i = 1, 2, \ldots, x\}$.

Next we form the tree of supersets. In this version of the algorithm each superset has a unique parent superset (unless it is the root of the tree), as it has exactly one vertex in its top level, and this vertex belongs to the bottom level of exactly one of the supersets of the higher layer. So in this version of the algorithm there is no need to build the tree and to rank its vertices in parallel, but rather we build the tree and then rank it in the standard way. (That is, the leaves are assigned rank 0. A non-leaf vertex v is assigned rank r where r is the maximum rank of its children if there is a unique child of v that has rank r; otherwise it is assigned rank $r + 1$.)

Now the algorithm of Section 4 for constructing the schedule is now applicable. It gives rise to a schedule of length

$$(d + O(\chi)) \cdot x + (d \cdot O(\log n) + O(\log^2 n)) \cdot \log n \ .$$

Set $x = \min\{Rad, \sqrt{\frac{Rad}{\chi}} \log n\}$. It follows that for $Rad < \frac{\log^2 n}{\chi}$, $x = Rad$ and the length of the schedule is $O(Rad \cdot \chi + \log^3 n) = O(\log^3 n)$. For $Rad \geq \frac{\log^2 n}{\chi}$ the length of the schedule is $Rad + O(\sqrt{Rad \cdot \chi} \cdot \log n + \log^3 n)$. Note that $Rad \geq \sqrt{Rad \cdot \chi} \cdot \log n$ whenever $Rad \geq \chi \log^2 n$. Hence for $Rad \geq \chi \log^2 n$ the schedule is of length $O(Rad + \log^3 n)$, and for $Rad < \chi \log^2 n$ it is of length $Rad + O(\chi \log^2 n + \log^3 n)$. Particularly, when $\chi = O(\log n)$ we get an upper bound of $O(Rad + \log^3 n)$ that is better than our bound of $O(Rad + \log^4 n)$ (that, however, applies to general graphs).

Graphs of genus g have arboricity $O(\sqrt{g})$, and graphs with arboricity a are $2a$-colorable. So graphs of genus g are $O(\sqrt{g})$-colorable. Observe that the genus of each graph $\tilde{G}_i$ is at most the genus g of the original graph G, and so $\chi = \max\{\chi_i \mid i = 1, 2, \ldots, x\} = O(\sqrt{g})$. For $Rad \geq \log^2 n$ the length of the schedule is at most $Rad + O(\sqrt{Rad} \cdot g^{1/4} \cdot \log n + \log^3 n)$. In particular, for graphs of constant genus (including planar graphs and graphs that exclude small minors) our upper bound is $Rad + O(\sqrt{Rad} \cdot \log n + \log^3 n) = O(Rad + \log^3 n)$.

Note that this upper bound is *existential* for graphs with small chromatic number, and we do not have a polynomial-time algorithm that achieves this bound because a direct implementation of our algorithm would require to color an arbitrary graph with a near-optimal number of colors. However, for graphs of bounded arboricity (such as *planar* graphs) our bound is *constructive*, because those graphs can obviously be efficiently colored with a near-optimal number of colors.

References

[1] N. Alon, A. Bar-Noy, N. Linial, and D. Peleg. A lower bound for radio broadcast. *Journal of Computer and System Sciences*, 43:290–298, 1991.

[2] B. Awerbuch, B. Berger, L. Cowen, and D. Peleg. Fast network decomposition. In *Proc. 11th ACM Symp. on Principles of Distr. Comp.*, pages 161–173, 1992.

[3] R. Bar-Yehuda, O. Goldreich, and A. Itai. Efficient emulation of single-hop radio network with collision detection on multi-hop radio network with no collision detection. *Distributed Computing*, 2(5):67–72, 1991.

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

[5] I. Chlamtac and S. Kutten. A spatial-reuse TDMA/FDMA for mobile multihop radio networks. In *Proc. IEEE INFOCOM*, pages 389–394, 1985.

[6] I. Chlamtac and Weinstein. The wave expansion approach to broadcasting in multihop radio networks. In *Proc. IEEE INFOCOM*, pages 874–881, 1987.

[7] B. Chlebus, L. Gasieniec, A. Gibbons, A. Pelc, and W. Rytter. Deterministic broadcasting in ad hoc radio networks. *Distributed Computing*, 1(15):27–38, 2002.

[8] M. Chrobak, L. Gasieniec, and W. Rytter. Fast broadcasting and gossiping in radio networks. In *Proc. 41st Symp. on Foundations of Computer Science*, pages 575–581, 2000.

[9] A. F. Clementi, A. Monti, and R. Silvestri. Distributed broadcast in radio networks of unknown topology. *TCS*, 3(302):337–364, 2003.

[10] A. Czumaj and W. Rytter. Broadcasting algorithms in radio networks with unknown topology. In *Proc. 44th Symp. on Foundations of Computer Science*, pages 492–501, 2003.

[11] M. Elkin and G. Kortsarz. Polylogarithmic additive inapproximability of the radio broadcast problem. In *Random-Approx 2004*, pages 105–116, 2004.

[12] I. Gaber and Y. Mansour. Broadcast in radio networks. In *Proc. of the 6th ACM-SIAM Symp. on Discr. Algorithms*, pages 577–585, 1995.

[13] P. Indyk. Explicit constructions of selectors and related combinatorial structures, with applications. In *Proc. 15th Symp. on Discr. Algorithms*, pages 697–704, 2002.

[14] D. Kowalski and A. Pelc. Deterministic broadcasting time in radio networks of unknown topology. In *Proc. 43th Symp. on Foundations of Computer Science*, pages 63–72, 2002.

[15] D. Kowalski and A. Pelc. Broadcasting in undirected ad hoc radio networks. In *Proc. 16th ACM Symp. on Principles of Distr. Comp.*, pages 73–82, 2003.

[16] D. Kowalski and A. Pelc. Centralized deterministic broadcasting in undirected multi-hop radio network. In *Random-Approx 2004*, pages 171–182, 2004.

[17] E. Kushilevitz and Y. Mansour. An $\omega(d\log(n/d))$ lower bound for broadcast in radio networks. *SIAM J. of Computing*, 27(3):702–712, 1998.

[18] N.Linial and M. Saks. Decomposing graphs into regions of small diameter. In *Proc. 2nd ACM-SIAM Symp. on Discr. Alg.*, pages 320–330, 1991.