Chapter 1
Broadcast in Radio Networks

Iris Gaber* Yishay Mansour†

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) 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.

1 Introduction

The importance of radio communication networks is increasing. The applications of radio communications has shifted from the traditional military applications to civilian applications such as cellular phones and wireless local area networks (LAN). One of the main advantages of radio communication is the the relative small investment in a rigid infrastructure. In addition, radio networks allows for mobile users, a feature which is crucial in many applications.

One of the basic tasks in a radio networks is broadcast. Broadcast is used for locating a user (e.g., in a cellular phone cell), checking that a station exits (e.g., in a wireless LAN), propagating topological information, etc. The unique property of radio communication is 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 question of generating a broadcast protocol for a radio network is well studied and various algorithms have been proposed. The common parameters are n, the number of nodes in the network, and D, the diameter of the network.

Centralized and distributed algorithms that compute a schedule for broadcast in a radio network are presented in [7, 8]. The time complexity is a function of the maximum node degree, which can be $\Omega(n)$, independent of D. Simulating a distributed protocol on a radio network is done in [2].

A centralize deterministic polynomial time algorithm for scheduling a bipartite graph in $O(log^2 n)$ time slots is given in [11]. They extend this result to a general network showing a schedule of $O(Dlog^2 n)$ time units.

The work of [6] gives a distributed randomized protocol for radio networks where a node has no topological knowledge of the network, even not about its neighbors, and the expected time of the broadcast is, $O(D \log n + \log^2 n)$.

A lower bound of $\Omega(D)$ follows simply since the message needs to reach the furthest node from the source. In [1] it is shown there exists a family of diameter-4 networks in which *any* broadcast schedule (even centralized) requires at least $\Omega(log^2 n)$ rounds of transmission. This result proves an $\Omega(D + log^2 n)$ lower bound, even when the graph is known.

The work of [12] derives a lower bound that depends on the diameter of the network. It was proved that for each *distributed* algorithm in which the topology of the network is not known in advance, there exists a graph for which the algorithm runs in time $O(D \log(n))$. Consequently, under the assumption that the topology of the network is unknown, the distributed algorithm presented in [6] is optimal.

In this work, we want to resolve the question whether for any graph there *exists* a schedule that runs in time $\Omega(D)$. From [1] we know that this cannot be true in general, since there are graphs of $O(1)$ diameter that requires $\Omega(\log^2 n)$ 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 $D < \log^2 n$ there are graphs which require $\Omega(\log^2 n)$ time. We show that for $D > \log^5 n$, there is always a schedule of time $O(D)$. The range of diameters between $\Omega(\log^2 n)$ and $O(\log^5 n)$ remains an open problem.

Our work uses the idea of *network partition*, introduced in [3], and used in many other applications (e.g., [4, 5]). We use the idea of partition within each group, where a group is a large number of consecutive BFS levels. Within each group we define clusters such that: (1) Each cluster has a relatively small diameter, (2) The union of the clusters covers the group, and, (3) The clusters' graph (obtained by treating each cluster as a

*Computer Science Dept., Tel Aviv University.
†Computer Science Dept., Tel Aviv University. e-mail: `mansour@cs.tau.ac.il`.

node) can be colored by $O(\log n)$ colors. (A node can participate in more than one cluster.)

Our algorithm computes for each node, its colors and type, where there are four different types of nodes. Given this information the broadcast can be performed from the source in time $O(D + \log^5 n)$. Note that this is a fairly small amount of additional information that is required in each node (i.e., only $O(logn)$ bits).

Our preprocessing algorithm is centralized and runs in polynomial time. (In fact it requires only $O(mn)$ time, where m is the number of edges and n is the number of nodes.) An interesting open question is how to perform this precomputation in a distributed way. Clearly, one can send all the topological information to the source and perform the computation there. If one can amortize over "many" broadcasts, then the communication cost of the precomputation would become negligible.

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 and give the nodes ranks. The main algorithm including the specific protocols for the nodes is presented in Section 6. In Appendix A we prove the complexity of the schedule and in Appendix B we compute the sequential complexity of the algorithm.

2 The Model

We model the multi-hop radio network as an undirected graph $G(V, E)$, where the 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 if exactly one of its neighbors transmits at that time slot. (If more than one neighbor transmits, then the processor does not receive any information.)

A *schedule* of a node tells 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 given message from a given node, called *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 graph $G(V, E)$ and a node $s \in V$ (which is called the *source*), and the output is a schedule that solves the broadcast problem, i.e., a schedule for each node such that at the end of the schedule each $v \in V$ will receive the message from the source s.

3 Algorithm Overview

As stated before we wish to construct a schedule for broadcasting a message from the source node to the rest of the nodes in the graph. Our algorithm has the following stages. First we divide the graph to x groups, according to their distance from the source, i.e., each group has D/x levels.

Our work uses the idea of *network partition* introduced in [3] within each group. For each group we define clusters such that (1) Each cluster has a relatively small diameter, (2) The union of the clusters covers the group, and, (3) The clusters' graph (obtained by treating each cluster as a node) can be colored by $O(\log n)$ colors.

The objective is to send the message to one node in each cluster, so that other nodes, which are close to such a recipient, can obtain the message from it (according to a known broadcast protocol). On the clusters' graph we build a directed spanning tree of depth x. 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 $log(n)$ edges connecting a node with a low-priority son. Also, the delay for the high priority is $O(D/x)$, thus the total delay of the high priority is $O(D)$.

Having set up the graph in this manner, we can now build a schedule which is composed of three processes. Two processes take place within clusters, both start at the first node that receives the message within the cluster (referred to as **messenger**), and run in parallel (they interleave in time). The third process is responsible for transferring the message between clusters. These three processes are performed alternately, on different clock ticks.

During the first process, called Broadcast-through, the message is propagated from the messenger to a node (referred to as the **chosen representative**) that forwards the message to another cluster – the one with the high priority. The path taken in this process consists of D/x nodes lying on consecutive BFS levels. The paths and the colors are chosen in such a way that (1) two paths of the same color cannot interfere one another, and (2) different colors can be pipelined so that they will not interfere one another. The total time complexity of the Broadcast-through process is $O(D)$. 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 $O(D/x \log^2 n)$.)

For the Broadcast-all process, we apply the algorithm of [6]. The main idea is that for any cluster, in order to reach it, we pass through at most $O(\log n)$ Broadcast-all processes. This is the main reason that one can force the time of the Broadcast-all process to be negligible in the overall cost, when the diameter is sufficiently large (i.e. $D = \Omega(\log^5 n)$ and $x = \log^3 n$).

4 Building the Clusters

A central notion in our algorithm is the notion of clusters. We divide the graph to clusters, which may be overlapping. Then the broadcast is done on the graph of clusters. We start with a few definitions.

DEFINITION 4.1. *We define all nodes with the same distance from the source as a* **BFS level**, *and we denote BFS level j by* $l_j = \{v \mid dist(source, v) = j\}$.

We divide the graph to groups of D/x levels, where $G_i = \{v \mid v \in l_j,\ (i-1)\cdot D/x + 1 \le j \le i \cdot D/x\}$. Given the group G_i, the **lowest** level of G_i is $l_{(i-1)D/x+1}$, and the **highest** level of G_i is $l_{i \cdot D/x}$.

Our algorithm of building the clusters, is applied to each group separately, and consists of two stages:

1. Defining sets in each group G_i. One set for each node in the lowest level of G_i.

2. Merging the sets into clusters that have the desired properties.

During the broadcast the goal of a cluster is to receive the message at the lowest level and pass it on to the highest level. Therefore, each cluster must have a node at the lowest level of the group, and each node must belong to at least one cluster. Our goal is to construct clusters that have a relatively small diameter, and that the clusters' graph can be colored by few colors. We can broadcast in parallel through clusters that are colored by the same color, for this reason we wish to color the clusters with few colors as possible, and thus achieve a high degree of pipelining. The small diameter guarantees that when we broadcast to all the nodes in the cluster, the time complexity is not too high.

Building such clusters is achieved by first defining sets, each set has a node in the lowest level, which is called the kernel, and all its neighbors at the successive level that satisfy the rule that the distance between a node to the kernel of the set is equal to the difference between their BFS numbers. This construction guarantees that each node at the highest level has an ancestor at the lowest level, such that the shortest path between them is contained within the set. However, we have no control on the level of overlapping between the sets. In the second stage we merge the sets into clusters that have the desired properties.

4.1 Defining the sets.

The sets of group G_i are defined as follows. For each node u in the lowest level of G_i, we add all the neighbors of u at the next level, and proceed from level l_h to its successive level l_{h+1}, by adding to the set all the nodes in level l_{h+1} that are neighbors of the nodes in the set in level l_h. Formally, each every $u \in l_{((i-1)\cdot D/x)+1}$ we define $S_u^{(i)}$ as follows, $S_u^{(i)} \stackrel{\text{def}}{=} \{v \mid v \in l_j \wedge dist(u,v) = j - ((i-1)\cdot D/x + 1),$ $((i-1)\cdot D/x) + 1 < j \le i \cdot D/x\}$

In other words, for each node u, in the **lowest** level of G_i we add all the nodes at distance D/x at the **highest** level (if u has such neighbors) and all the nodes on the paths from u to them. Looking at the group as a directed graph, in which the edges are all directed from a lower level to a higher level, a set is the connected subgraph whose root is u.

4.2 Constructing the clusters.

Since the sets in a group may overlap, it is quite possible that broadcasts within one set will cause interference in another. In our next step, we are going to build clusters, composed of unions of sets. The clusters will be built so that they can be colored such that no two clusters that belong to the same color interfere with each other's broadcasts.

The benefit from such a coloring is that broadcast made within one cluster, C_r does not disturb any broadcasts made in other clusters of the same color, since there are no edges connecting a node of C_m to a node of C_r. Note that there can be overlapping between the clusters, however two clusters that overlap are colored with different colors.

DEFINITION 4.2. *A* **coloring** *of clusters* $C_1, \ldots, C_m$ *with* α *colors is an assignment of a color (from* $\{1, \ldots, \alpha\}$*) to each* **cluster** *so that there is no edge connecting any two clusters of the same color, i.e.,* $\forall v_i \in C_i, v_j \in C_j\ i \ne j$ *if* $(v_i, v_j) \in E$ *then* $color(C_i) \ne color(C_j)$

When building the clusters, we have two major goals. The first is to get 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. The second goal is to be able to color the clusters with a minimum number of colors. By coloring the clusters with a small number of colors we achieve the ability to make many broadcasts in parallel - recall clusters of the same color can broadcast in parallel with no collisions.

THEOREM 4.1. *Given the sets :* $S_1^{(i)}, \cdots S_m^{(i)}$ *there are clusters* $C_1^{(i)}, \cdots C_k^{(i)}$ *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 diameter of each cluster* C_j *is at most*

$O(D/x \cdot log(n))$, i.e.,
$\forall u, v \in C_j^{(i)} \quad dist(u,v) = O(D/x \cdot log(n))$.

4. *There is an α coloring of the clusters C_j with $\alpha = O(log(n))$ colors, and*
5. *The clusters can be constructed from the sets in $O(mlog(n))$ time.*

Proof. The proof uses a variant of the algorithm of [3], and is omitted from this extended abstract. □

DEFINITION 4.3. *For two nodes, $u \in l_i$ and $v \in l_j$ let* **path(u,v)** *be the 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 the path consists of $j - i$ nodes. Notice that a path does not always exist and it is not necessarily unique.

LEMMA 4.1. *Consider a group G_i, and a cluster $C_j^{(i)}$. For each node $v \in C_j^{(i)}$, at the highest level, there is a node $u \in C_j^{(i)}$, at the lowest level such that there exists a path(u,v) for which $path(u,v) \subseteq C_j^{(i)}$. i.e.,*

$\forall v \in l_{i \cdot D/x} \cap C_j^{(i)} \ \exists u \in l_{(i-1)\cdot D/x+1} \cap C_j^{(i)}$ *s.t.* $dist_{C_j^{(i)}}(u,v) = D/x \ \wedge \ path(u,v) \subseteq C_j^{(i)}$.

(We refer to such a u as the **ancestor** *of v.)*

Proof. The cluster, $C_j^{(i)}$ is a union of sets $S_1^{(i)}, \ldots, S_m^{(i)}$. If $v \in C_j^{(i)}$, then there exists a set, $S_r^{(i)}$, such that $v \in S_r^{(i)}$.

The set $S_r^{(i)}$ was built from some kernel node u, where u lies at the lowest level of the group G_i, i.e., level $l_{(i-1)\cdot D/x+1}$. Node v is at distance D/x from u, and the entire path from u to v is in $S_r^{(i)}$, and therefore in $C_j^{(i)}$. □

5 Building the Clusters Graph

After building the clusters in all the groups, we proceed to define the clusters' graph.

The general idea is to pass the message between the clusters. Each cluster chooses a *sender* - a unique cluster in the preceding group - from which it will receive the message.

The procedure of picking the senders is applied to the clusters of each group, starting from the clusters of the last one, G_x, and continuing towards the clusters of the first group, 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 view the clusters-graph as a directed tree where an edge from $C_j^{(i)}$ to $C_r^{(i+1)}$ indicates that the message is to be passed from $C_j^{(i)}$ to $C_r^{(i+1)}$.

5.1 Choosing the Sender. The procedure of picking a sender for a cluster $C_j^{(i)}$ consists of two stages. At the first stage the cluster notes the clusters that have chosen it as their sender, and according to a ranking algorithm finds the one to which the message should get first (the one with the highest rank). At the second stage, according to the node chosen in the first stage, it picks its sender from the set of clusters $C_k^{(i-1)}$ to which it is connected. (It computes an ancestor of the node, and then chooses a cluster to which the ancestor is connected.)

For each cluster $C_j^{(i)}$, in group G_i we choose a vertex u_i, at its lowest level which is its **messenger**. The goal of the messenger is to receive the message from some vertex v_{i-1}, in a cluster of the previous group G_{i-1}, and start broadcasting it in $C_j^{(i)}$. The vertex v_{i-1} lies at the highest level of a cluster in G_{i-1} and is called a **representative** (there is such a node since we used BFS numbers).

A cluster $C_j^{(i)}$ whose messenger is supposed to receive the message from a **representative** at cluster $C_k^{(i-1)}$ is considered to be a **neighbor** (also, a **son**) of $C_k^{(i-1)}$ and we add a directed edge from $C_k^{(i-1)}$ to $C_j^{(i)}$.

The stages of picking the sender are:

1. The cluster $C_j^{(i)}$ determines its rank according to the rank of its neighbors which are clusters in the previous group. (The ranking procedure is described below.)

2. The cluster picks the messenger from the nodes of its lowest level. The messenger is connected to the cluster with the highest rank.

3. The cluster picks the representative, i.e. a node at the preceding group that will forward the message to its messenger.

5.2 The Ranking Procedure. Note that the edges between the representative and the messenger create a tree structure between the clusters. The definition of a rank in a tree is the standard definition (see [9].) A cluster that is a leaf has a rank of 0. Otherwise, assuming $r = \max\{rank(T) \mid T \in neighbors(C)\}$ a cluster C gets its rank as follows:

$$rank(C) = \begin{cases} r+1 & \text{if } \mid \{T \mid rank(T) = r\} \mid > 1 \\ r & \text{otherwise} \end{cases}$$

The following is a simple and well known fact about the rank of a tree.

LEMMA 5.1. *For any tree T with n nodes: $0 \leq rank(T) \leq log(n)$.*

5.3 Picking the Messenger. If a cluster C in group G_i has a neighboring cluster F in G_{i+1}, such that $rank(C) = rank(F)$, the node $v \in C$ which is the representative of F will be called the **chosen representative**. The **messenger** of C will be the *ancestor* of the chosen representative. (By Lemma 4.1, such a node exists in C.)

If C has no such neighboring cluster F, the messenger will be chosen arbitrarily from the nodes of the lowest level.

5.4 Picking the representative. 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 group, G_i, i.e., in level $l_{(i-1)\cdot D/x+1}$. Therefore its representative lies at the highest level of the previous group, G_{i-1}, i.e. in level $l_{(i-1)\cdot D/x}$.

See Figure 1 for an example of a clusters' graph.

6 The main algorithm

The broadcasting starts at the sender which is part of the first group, G_1. At G_1 there is only one cluster, C since the lowest level contains only one node, the source. 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 group 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 the neighboring clusters, $C_j^{(i+1)}$. The message advances in two ways. A fast way is from the messenger to the chosen representative and a slow way where it is being broadcast to all the nodes. The two ways are interleaved and do not interfere with each other.

6.1 The Procedure for a Single Cluster. Each cluster, when the messenger receives the message, performs the three following tasks:

- The message is passed along the path from the messenger to the chosen representative. Recall that the chosen representative is unique, and the path is within the cluster.

- The message is broadcast to all the nodes including the other representatives in the cluster.

- A representatives, upon receiving the message, broadcast it to the messenger (which is in the next group.)

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 $t \equiv 0 \pmod 3$, and is used to forward the message to the representative. The **Broadcast-All** clock is active at $t \equiv 1 \pmod 3$, and using it the messenger passes the message to all the cluster's nodes. The **Group-to-Group** clock is ative at $t \equiv 2 \pmod 3$, and using it the representatives pass the message to the messengers of the next group.

6.2 Updating the representative. Suppose a cluster's color is j ($1 \le j \le log(n)$). When the messenger of a cluster (whose color is j, $1 \le j \le log(n)$) gets the message, it waits until the broadcast-through clock has $time \equiv 3 \cdot j \pmod{3 \cdot \lceil 2log(n) \rceil}$ and then starts passing the message along the path to the representative. Each node on that path, upon receiving the message, passes it on to the next node on the path. (Recall that this is done using the broadcast through clock.)

6.3 Broadcasting to the rest of the cluster. The broadcasting to all the nodes of the cluster will be done using the algorithm **Anonymous Broadcast**, which is the randomized algorithm of [6].

THEOREM 6.1. *Given a graph,G, with n nodes and diameter D there exists a (randomized) scheduling for broadcasting in time $K \cdot (D \cdot log(n) + log^2(n))$, where K is a constant.*

Proof. The randomized algorithm of [6] has expected time of $O(D \cdot log(n) + log^2(n))$. Therefore, for some choice of coins, there is a scheduling that takes time $K \cdot (D \cdot log(n) + log^2(n))$, for some constant K. □

When broadcasting to all the nodes of the cluster, regardless of the color, the messenger waits until $time \equiv 0 \pmod{K \cdot D/x \cdot log^2(n)}$ where K is the constant of Theorem 6.1. Then, along with the rest of the nodes at that level, it applies the **Anonymous Broadcast** algorithm where $D' = O(D/x \cdot log(n'))$. Since $n' < n$ after at most $K \cdot D/x \cdot log^2(n)$ clock ticks all the representatives at level $l_{i \cdot D/x}$ receive the message. (Recall this is done using the broadcast-all clock.)

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

6.5 Merging the directions. When a messenger u of a cluster $C_j^{(i)}$ gets the message, it initiates the first two algorithms: the algorithm for forwarding on the path to the chosen representative (during the Broadcast-Through clock), and the algorithm for anonymous broadcasting (during the Broadcast-All clock).

Any other node which gets the message also acts according to the current context. When a node, which is on the path from a messenger to a representative, at some level l_k receives the message, it waits for the broadcast-through clock to be active, and then forwards the message on to the next node on the path, i.e. a node at level l_{k+1}. On the ticks of the broadcast-all clock, the nodes perform the Anonymous Broadcast algorithm.

When a representative v of cluster $C_j^{(i)}$ gets the message, it initiates the broadcasting to the next group algorithm, at the Group-to-Group clock ticks (in addition to the Anonymous Broadcast algorithm, at the Broadcast All clock ticks).

6.6 The Data Structure in a Single Node. When a node receives the message, it acts according to a combination of its type and color in the context of the cluster that has sent the message.

Since a node can belong to many clusters, it should know where the message came from, and play according to its role in that cluster. A node can not, however, belong to different clusters of the same color (this would imply that those clusters are neighbors and have the same color - in contradiction to the construction), therefore, it can belong to a maximum of $log(n)$ clusters.

In order to represent its types, a node holds a vector of $log(n)$ entries - an entry for each color. Each such an entry represents whether the node belongs to a cluster of that color, and if it does - what is its type.

In a cluster, a node can be one of the following types: (1) A messenger node, (2) A representative, (3) A node on the path from the messenger to the chosen representative, and (4) Any other node.

In the context of the Broadcast-All algorithm, all nodes behave according to the algorithm of [6], regardless of their color or type.

In the context of the Broadcast-Through algorithm, a node may act in one of the following ways: (1) A messenger node waits for the turn of its color to start broadcasting, (2) A node on the path from the messenger to the representative broadcasts immediately, (3) A node outside the path from the messenger to the representative ignores the message and remain silent.

In the context of the Group-to-Group algorithm a representative node waits for the next round of the broadcast-to-next-group algorithm and then applies it.

Recall that in the context of the Broadcast-Through algorithm, a node first verifies where the message has come from. There are two possible ways of doing it: The messages can also carry the color, starting from the messenger. (A messenger node always belong to one cluster only). Another way is calculating the source of the message, given the nodes level and the time of the receipt of the message.

References

[1] Noga Alon, Amotz Bar-Noy, Nati Linial, and David Peleg. A lower bound for radio broadcast. In *Journal of Computer and System Sciences 43*, pages 290–298, 1991.

[2] Noga Alon, Amotz Bar-Noy, Nati Linial, and David Peleg. Single round simulation on radio networks. In *Journal of Algorithms, 13*, pages 188–210, 1992.

[3] Baruch Awerbuch and David Peleg. Sparse partitions. In *31st Annual Symposium on Foundations of Computer Science*, pages 503–513, 1990.

[4] Baruch Awerbuch and David Peleg. Synchronization with polylogarithmic overload. In *31st Annual Symposium on Foundations of Computer Science*, pages 514–522, 1990.

[5] Y. Afek and M. Ricklin. Sparser: A paradigm for running distributed algorithms. In *Journal of Algorithms*, volume 14, pages 316–328, 1993.

[6] Reuven Bar-Yehuda, Oded Goldreich, and Alon Itai. On the time-complexity of broadcast in radio networks: An exponential gap between determinism and randomization. In *Proceedings of the 6^{th} Annual ACM Symposium on Principles of Distributed Computing*, pages 98–108. ACM, August 1987.

[7] I. Chlamtac and S. Kutten. An efficient broadcast protocol embedded in multi-hop radio networks. In *COMPCON Conf. Proceedings*, Fall 1984.

[8] I. Chlamtac and S. Kutten. On broadcasting in radio networks - problem analysis and protocol design. In *IEEE Trans. on Comm.*, page Vol 33 No. 12, Dec. 1985.

[9] Thomas H. Cormen, Charles E. Leiserson, and Ronald L. Rivest. *Algorithms.* MIT Press, 1990.

[10] I. Chlamtac and O. Weinstein. Distributed 'wave' broadcasting in mobile multi-hop radio networks. In *INFOCOM*, pages 82–89, 1987.

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

[12] E. Kushilevitz and Y. Mansour. An $\Omega(D \log n)$ lower bound for broadcast in radio networks. In *Proceedings of the 12^{th} Annual ACM Symposium on Principles of Distributed Computing, Ithaca, New York*, pages 65–73, 1993.

A Proof of Correctness and Complexity

We begin by studying the time taken for a message to arrive at a cluster's nodes and the representative in particular since the messenger of that cluster received it for the first time. The three procedures: "Updating the representative", "Broadcasting to the rest of the cluster" and "Broadcasting to the next group" are performed at the different clocks and do not interfere one with the other, at a given cluster. We can therefore examine the complexity of each one separately. Then, we proceed to investigate the time taken for a message to get from one cluster representative, to the messenger of its son-cluster. Finally, we prove that any node gets the message after no longer that $O(D + x \cdot \log^2(n) + D/x \cdot \log^3(n))$ clock ticks.

LEMMA A.1. *At a given time t of the broadcast-through clock at a given level l_k, of a cluster C in group G_i, there is only one color which is broadcasting and it does not collide with any other broadcasts of different colors.*

Proof. Each color j starts broadcasting at time $t_0 = 3 \cdot j \pmod{3 \cdot \lceil 2log(n) \rceil}$ of the broadcast-through clock. By definition, the path from the messenger to the chosen representative is a shortest path, therefore the message advances from a level l_h to a level l_{h+1} every broadcast-through clock tick and arrives at level l_k at time $t_0 + k - ((i-1) \cdot D/x + 1)$, where t_0 is the time it arrives to the messenger. By Lemma 4.1 the path is within the cluster C. A vertex of a different color, j' can be, at that time, broadcasting at level $l_{k'}$ where $\mid k - k' \mid \geq 3$. Since broadcasting at a level l_k can only get to a vertex at the nearest level, i.e. l_{k+1} or l_{k-1}, and $k' + 1 < k - 1$ or $k + 1 < k' - 1$ the broadcasting of the nodes of colors j and j' do not affect one another. □

LEMMA A.2. *Consider a group, G_i, $1 \leq i \leq x$, with clusters $C_1^{(i)}, \ldots, C_m^{(i)}$. Suppose the messenger of a cluster $C_j^{(i)}$ gets the message at time t_j. The chosen representative of cluster $C_j^{(i)}$, receives the message by time $t_j + D/x + O(\log(n))$ and the rest of the nodes (including the other representatives) receive it by time $t_j + O(D/x \cdot \log^2(n))$.*

Proof. The proof consists of two parts, according to the paths at which the message passes, and the nodes to which it is supposed to arrive. We consider the cluster $C_j^{(i)}$, whose color is α, whose messenger gets the message at time t_j.

- *Broadcast-Through pass* - broadcasting to the chosen representative.
- *Broadcast-All pass* - broadcasting to the rest of the cluster (including the other representatives).

<u>*Broadcast-through pass*</u>

On the broadcast-through clock ticks, the messenger passes the message to the next node on the path to the representative. First the messenger waits until the time becomes $3 \cdot \alpha \pmod{3 \cdot \lceil 2log(n) \rceil}$ (which is at most $3 \cdot \lceil 2log(n) \rceil$ clock ticks) and then the message is sent along the *path* to the representative. We claim that it does not collide with any other broadcast, and hence advances a node each Broadcast Through clock tick.

- From lemma A.1 the broadcasts of color α are not disturbed by other broadcasts of a different color.
- From the definition of coloring it follows that broadcasting in a cluster $C_k^{(i)}$ of color α does not interfere with any other broadcast at a cluster $C_j^{(i)}$ of the color α.

This implies that once the message is started in the broadcast-through clock it advances each broadcast-through clock tick to the successive level. The path from the messenger to the chosen representative is of length D/x as shown in lemma 4.1, therefore it gets to its destination after D/x broadcast-through clock ticks. Consequently, the broadcast-through passes take at most $3 \cdot log(n) + D/x$ Broadcast Through clock ticks.

<u>*Broadcast-all pass*</u>

On the broadcast-all clock ticks we wish to broadcast the message to all nodes of $C_j^{(i)}$. The messenger first waits until the time becomes $0 \pmod{K \cdot \lceil log^2(n) \rceil \cdot D/x}$ and then, (along with other nodes at its level) broadcasts the message to all nodes at distance $D/x \cdot \lceil log(n) \rceil$ from it. All messengers who get the message at the same $K \cdot \lceil log^2(n) \rceil \cdot D/x$ time interval as the messenger of $C_j^{(i)}$, wait at most $K \cdot \lceil log^2(n) \rceil \cdot D/x$ Broadcast All clock ticks before they start broadcasting, and then run together the Anonymous Broadcast algorithm. By theorem 6.1 it follows that after at most $K \cdot \lceil log^2(n) \rceil \cdot D/x$ Broadcast All clock ticks all nodes get the message and therefore two runs of the algorithm never interfere with one another as one run starts after the previous has ended. Consequently, the broadcast-all passes take at most $O(log^2(n) \cdot D/x)$. □

LEMMA A.3. *Consider two groups 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 $0 \pmod{K \cdot \log^2(n)}$ then it starts broadcasting using the Anonymous Broadcast algorithm.

By Theorem 6.1 it follows that by $K \cdot log^2(n)$ 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 algorithms. Therefore, u_{i+1} gets the message after at most $2K \cdot log^2(n)$. □

THEOREM A.1. *A vertex waits no longer than $O(D + x \cdot log^2(n) + D/x \cdot log^3(n))$ before it gets the message.*

Proof. A path from the source to a vertex contain "broadcast-through passes", "broadcast-all passes" and "group to group transfer". From the fact the a rank can be no more than $log(n)$ (Lemma 5.1) it follows that there are at most $log(n)$ "broadcast-all" passes on the path to v's cluster.

Lemma A.2 proves that each "broadcast-through" pass takes at most $O(D/x + log^2(n))$ time and each "broadcast-all" pass takes at most $O(D/x \cdot log^2(n))$ time. Lemma A.3 proves that each group to group transfer takes $O(log^2(n))$ time.

There are at most x such transfers, therefore, the total broadcast-through procedure takes $O(D+xlog^2(n))$, and the group to group transfer takes $O(x \log^2(n))$. Since there are at most $log(n)$ broadcast-all passes, its time is at most $O(D/x \cdot log^3(n))$. The total time is at most $O(D + x \cdot log^2(n) + D/x \cdot log^3(n))$. □

THEOREM A.2. *A vertex waits no longer than $O(D + log^5(n))$ before it gets the message.*

Proof. Let $x = log^3(n)$ in theorem A.1. □

B The Sequential Complexity of the Algorithm

LEMMA B.1. *The preprocessing phase takes $O(m \cdot n)$ time, where n is the number of nodes and m the number of edges.*

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. This procedure takes $O(m)$ time.

In the next stage we define the sets. Each node whose BFS number is a multiple of D/x becomes a kernel for a set. The algorithm adds to the set 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 set induces a sub-graph whose edges connect nodes that lie on different BFS levels in the original graph. Notice the number of sets is bound by n, and at each set the number of edges is bound by m. Therefore, the total complexity of this stage is $O(n \cdot m)$.

The construction of the clusters from the set consists of $O(log(n))$ iterations of a procedure named Part-Sep. which takes $O(m)$ time and therefore the total time of this stage is $O(log(n) \cdot m)$.

Having obtained the clusters' graph, we find an arbitrary spanning tree for it, this takes $O(m)$ time.

The last procedure is that of assigning ranks to the clusters, and picking a sender for each cluster accordingly. For each cluster, we should compute the rank for each of its neighboring sons. Since the graph is already a tree, the number of edges is n and therefore the complexity of this task is $O(n)$.

The total time for the algorithm is $O(m + n \cdot m + log(n) \cdot m + m + n) = O(n \cdot m)$. □

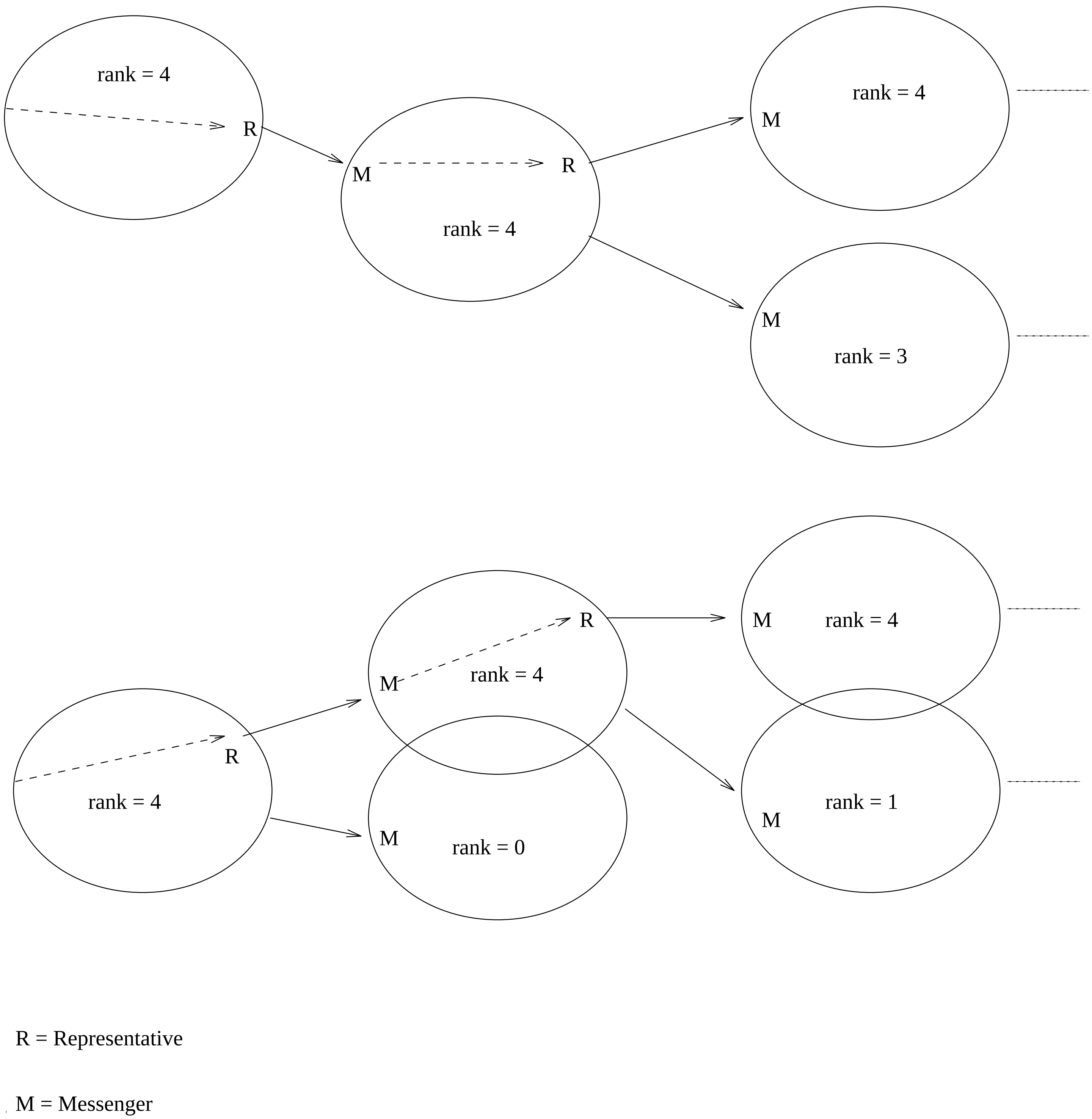

Figure 1: The clusters' graph