Coloring Unstructured Radio Networks

Thomas Moscibroda
Computer Engineering and
Networks Laboratory, ETH Zurich
8092 Zurich, Switzerland
moscitho@tik.ee.ethz.ch

Roger Wattenhofer
Computer Engineering and
Networks Laboratory, ETH Zurich
8092 Zurich, Switzerland
wattenhofer@tik.ee.ethz.ch

ABSTRACT

During and immediately after their deployment, ad hoc and sensor networks lack an efficient communication scheme rendering even the most basic network coordination problems difficult. Before any reasonable communication can take place, nodes must come up with an initial structure that can serve as a foundation for more sophisticated algorithms. In this paper, we consider the problem of obtaining a vertex coloring as such an initial structure. We propose an algorithm that works under the unstructured radio network model. This model captures the characteristics of newly deployed ad hoc and sensor networks, i.e. asynchronous wake-up, no collision-detection, and scarce knowledge about the network topology. Our algorithm produces a correct coloring with $O(\Delta)$ colors in time $O(\Delta \log n)$ with high probability in a unit disk graph, where n and Δ are the number of nodes in the network and the maximum degree, respectively. Furthermore, the number of locally used colors depends only on the local node density.

Categories and Subject Descriptors

F.2.2 [**Analysis of Algorithms and Problem Complexity**]: Nonnumerical Algorithms and Problems—*computations on discrete structures*;
G.2.2 [**Discrete Mathematics**]: Graph Theory—*graph algorithms*;
G.2.2 [**Discrete Mathematics**]: Graph Theory—*network problems*

General Terms

Algorithms, Theory

Keywords

coloring, radio network, initialization, asynchronous wake-up, ad hoc networks, sensor networks

1. INTRODUCTION

Wireless multi-hop radio networks such as ad hoc or sensor networks [1] are formed of autonomous nodes communicating via radio. Typically, if two nodes are not within their mutual transmission range, they may communicate through intermediate nodes. In other words, the communication infrastructure must be organized by the nodes themselves, rather than being provided as part of a fixed built-in infrastructure as in traditional wired networks.

The lack of available a-priori infrastructure is particularly pronounced during and after the deployment, when the network is unstructured and chaotic [14, 15, 19]. Before any reasonable communication can be carried out and before the network can start performing its intended task, the nodes must establish some kind of structure that allows an efficient communication scheme. Once this *initial structure* is achieved, sophisticated and well-studied algorithms and network organization protocols may be used on top of it. Naturally, the inherent problem faced when setting up such an initial structure is that there is no existing infrastructure that could facilitate the task. In fact, coping with the absence of an *initial structure* is one of the quintessential tasks in ad hoc and sensor networks and finding efficient solutions for that purpose is of great practical importance. In existing systems such as Bluetooth, for instance, the initialization tends to be slow even for a small number of devices.

In this paper, we study the construction of an initial structure useful for subsequent network organization tasks. Technically, we study how the network nodes can quickly compute a good *vertex coloring* without relying on any existing infrastructure. A correct vertex coloring for a graph $G = (V, E)$ is an assignment of a color $color(v)$ to each node $v \in V$, such that any two adjacent nodes have a different color. The importance of a vertex coloring as an initial structure in wireless ad hoc and sensor networks is well-motivated because associating different colors with different time-slots in a time-division multiple access (TDMA) scheme; a correct coloring corresponds to a medium access control (MAC) layer without *direct interference*, that is, no two neighboring nodes send at the same time.

It is well known that in order to guarantee an entirely collision-free schedule in wireless networks, a correct vertex coloring is not sufficient, for what is needed is a coloring of the *square* of the graph, i.e., a valid distance 2-coloring [13, 26]. However, besides being a non-trivial first step towards obtaining a distance 2-coloring, a simple vertex-coloring ensures a schedule in which a receiver can be disturbed by at most (a small) constant number of interfering senders in a given time-slot. This allows for simple randomized algorithms guaranteeing every sender a constant sending probability in each scheduled time-slot. As the available bandwidth (and hence the possible throughput) of a node v in such a schedule depends on the highest color assigned in its local 2-neighborhood, only low colors should be assigned in sparse areas of the network, whereas the higher colors should only be used in dense areas. Particularly, a good coloring should have the property that the highest color assigned to a node in each neighborhood should depend only on the *local node density* of that neighborhood.

In view of our goal of setting up an initial MAC scheme in a newly deployed network, our coloring algorithm must not rely on any previously established MAC layer. Instead, we are interested in a simple algorithm that quickly computes a coloring entirely from *scratch.* Note that this precludes algorithms working under any sort of *message passing model* in which nodes know their neighbors a-priori, and in which messages can be sent to neighbors without fearing collision, e.g. [3, 7, 24]. Studying classic network coordination problems such as coloring in absence of an established MAC layer highlights the *chicken-and-egg* problem of the initialization phase [14]. A MAC layer ("chicken") helps achieving a coloring ("egg"), and vice versa. The problem is that in a newly deployed ad-hoc/sensor network, there is typically no built-in structure, i.e. there are neither "chickens" nor "eggs."

Clearly, one important aspect when studying the initialization phase of ad hoc/sensor networks is to use an appropriate model. On the one hand, the model should be realistic enough to actually capture the particularly harsh characteristics of the deployment phase. But on the other hand, it ought to be concise enough to allow for stringent reasoning and proofs. Recently, the *unstructured radio network model* has been proposed as a model that attempts to combine both of these contradictory aims [14]. It makes the following assumptions.

- We consider *multi-hop* networks, that is, there exist nodes that are not within their mutual transmission range. Therefore, it may occur that some neighbors of a sending node receive a message, while others experience interference from other senders and do not receive the message. The wireless nature of the communication graph is modeled as a unit disk graph in which two nodes are neighbors iff their Euclidean distance is at most one.

- Nodes can wake up *asynchronously.* In a wireless, multi-hop environment, it is realistic to assume that some nodes wake up (e.g. become deployed, or switched on) later than others. Thus, nodes do not have access to a global clock. Contrary to work on the so-called *wake-up problem* [5, 12], nodes are *not* woken up by incoming messages, that is, sleeping nodes do neither send nor receive any messages. Finally, the node's wake-up pattern can be completely arbitrary.

- Nodes do not feature a reliable *collision detection* mechanism. This assumption is often realistic, considering that nodes may be tiny sensors [1] with equipment restricted to the minimum due to limitations in energy consumption, weight, or cost. Moreover, the sending node itself does not have a collision detection mechanism either. Hence, a sender does not know how many (if any at all!) neighbors have received its transmission correctly.

- At the time of their waking-up, nodes have only limited knowledge about the total number of nodes in the network and no knowledge about the nodes' distribution or wake-up pattern. Particularly, they have no a-priori information about the number of neighbors and when waking up, they do not know how many neighbors have already started executing the algorithm.

Naturally, algorithms for such uninitialized, chaotic networks have a different flavor compared to "traditional" algorithms that operate on a given network graph that is static and well-known to all nodes.

In this paper, we show that even in this restricted model, a good vertex coloring can be computed efficiently. Specifically, we propose a randomized algorithm that computes a correct vertex coloring using $O(\Delta)$ colors in time $O(\Delta \log n)$ with high probability. Furthermore, our algorithm features the property that the highest color assigned to any node in a certain area of the network depends only on the *local density* of that area.

To get a feel for the algorithm's efficiency, consider a network with maximum degree Δ. In a unit disk graph, any correct coloring requires $O(\Delta)$ colors. Also, a node having Δ neighbors must receive a message from all its neighbors in order to make sure that its color does not collide with any color chosen by one of its neighbors. Because all messages must arrive *without collision*, this process takes at least $\Omega(\Delta)$ time-slots. This implies a $\Omega(\Delta)$ lower bound for our coloring problem.

The remainder of the paper is organized as follows. An overview of related work is given in Section 2. Section 3 introduces our model of computation in detail. Our coloring algorithm is subsequently presented and analyzed in Sections 4 and 5. Finally, Section 6 concludes the paper.

2. RELATED WORK

Coloring graphs belongs to the most fundamental NP-hard problem in theoretical computer science and has been thoroughly studied. In distributed computing, the study of vertex coloring has lead to several seminal contributions. Cole and Vishkin gave a deterministic distributed algorithm for computing a correct coloring on a ring using three colors in time $O(\log^* n)$ [3]. A generalization of the same technique can be used to color trees and arbitrary bounded-degree graphs with 3 and $\Delta + 1$ colors in time $O(\log^* n)$, respectively [7]. All these upper bounds were subsequently shown to be tight by Linial [16], even for the case of randomized algorithms. For arbitrary graphs, a $\Delta + 1$-coloring can be computed in time $O(\log^* n + \Delta^2)$ [24] or $O(\Delta \log n)$ [6]. Moreover, [18] claims that it is possible to compute a $O(\Delta)$-coloring in time $O(\log^* n)$. The authors of [8] present distributed approaches for finding colorings in graphs that admit a coloring with less than $\Delta + 1$ colors. Finally, an experimental study of various vertex coloring algorithms is given in [4].

All the above algorithms are based on a *message passing model* [25] that abstracts away problems such as interference, collisions, asynchronity, or the hidden-terminal problem. Specifically, it is assumed that nodes know their neighbors at the beginning of the algorithm and that the transmission of messages is handled flawlessly by an existing, underlying MAC layer. Furthermore, all nodes wake up synchronously and start the algorithm at the same time. As motivated in the introduction, these assumptions are invalid when studying multi-hop radio networks during or immediately after their deployment.

In view of its practical importance, it is not surprising that there has recently been a lot of effort in designing efficient algorithms for setting up initial structures, i.e., [9, 28, 19, 15]. The *unstructured radio network* model was first proposed in [14] and subsequently improved and generalized in [15]. It is an adaptation of the classic *radio network model* (e.g., [2]), combining various of its flavors in order to model the harsh conditions during and immediately after the deployment. [15] proposes an algorithm that efficiently computes a *minimum dominating set* approximation from scratch. The paper [20] goes one step further by giving an algorithm for computing a *maximal independent set* in the unstructured radio network model in time $O(\log^3 n/\log\log n)$. Notice that in contrast to our paper, all of the above results require *three independent communication channels*. In [21], the above mentioned MIS algorithm was improved to $O(\log^2 n)$ using only a single communication channel.

Coloring networks for the purpose of obtaining a channel assignments or TDMA scheme has been studied in [13, 26], among others. Moreover, coloring a network in which all nodes are within mutual transmission range of all other nodes (*single-hop* networks) reduces to the so-called *initialization problem*. This problem has been feverously studied and analyzed during the past years [22, 23]. For several reasons, the approach taken in these papers cannot be translated into efficient algorithms for the unstructured radio network model. First and foremost, the multi-hop character of our network model complicates matters. In the single-hop case, if there is a collision, no node in the network receives a message, whereas in our multi-hop scenario, it is likely that some neighbors of the sender may receive the message, while others experience a collision and do not receive the message. This difference renders it impossible for nodes to keep a coherent picture of the local situation. Secondly, most initialization papers assume *strong communication* [11], that is, a sending node can hear whether its message was successfully received by all nodes or whether it has caused a collision. In a multi-hop scenario, this assumption does not make sense. Finally, unlike [22, 23], we consider asynchronous wake-up where nodes can wake up at any time.

There has also been work on models containing asynchronous wake-up. In the so-called *wake-up problem* [5, 12], the goal is to wake up all nodes in the graph as quickly as possible by sending them messages. The assumption made in these papers is that a node is *woken up* by an incoming message. While this leads to interesting algorithmic problems, it does not reflect the reality in newly deployed ad hoc or sensor networks.

3. MODEL AND NOTATION

In the *unstructured radio network model* [14], we consider *multi-hop* radio networks *without collision detection*. That is, nodes are unable to distinguish between the situation in which two or more neighbors are sending and the situation in which no neighbor is sending. A node receives a message if and only if exactly one of its neighbors sends a message. Nodes may wake up *asynchronously* at any time.

Upon wake-up, a node has no information as to whether it is the first to wake up, or whether other nodes have been running the algorithm for a long time already. We call a node *sleeping* before its wake-up, and *awake* thereafter. Only awake nodes can send or receive messages, and sleeping nodes are *not* woken up by incoming messages. The two extreme cases of our asynchronous wake-up model are the following. First, all nodes start synchronously at the same time, or only one of the sleeping nodes wakes up while all others remain sleeping for a long time. Recall again that nodes are unaware which (if any) of the two extreme cases holds. The *time complexity* T^i of a node v_i is defined as the number of time-slots between the node's *waking up* and the time it has made its irrevocable *final decision* on its color. The algorithm's *time complexity* is the maximum number T^i over all nodes in the network.

As customary, we consider *Unit Disk Graphs* (UDG) to model the wireless multi-hop network. In a UDG $G = (V, E)$, there is an edge $(u, v) \in E$ iff the Euclidean distance between u and v is at most 1. Note that due to asynchronous wake-up, some nodes may still be asleep, while others are already sending. Hence, at any time, there may be sleeping nodes which do *not receive* a message in spite of their being within the transmission range of the sender. When waking up, nodes have only scarce knowledge about the network graph's topology. In particular, a node has no information on the number of nodes in its neighborhood. However, every node has estimates n and Δ for the number of nodes in the network and the maximum degree, respectively. In reality, it may not be possible to foresee these global parameters precisely by the time of the deployment, but it is usually possible to pre-estimate rough bounds.

For the sake of simplicity, we assume time to be divided into discrete time-slots that are synchronized between all nodes. This assumption is used merely for the purpose of facilitating the analysis, i.e., our algorithm does not rely on this assumption in any way (see the standard argument given in [27]), as long as each node's internal clock runs at the same speed.

In each time-slot, a node can either send or not send. If a node sends in a time-slot t, it does not receive any messages in time-slot t. A node v receives a message in a time-slot t only if *exactly one node* in its neighborhood sends a message in this time-slot (and if it is not sending itself). The message size in our model is limited to $O(\log n)$ bits per message. Further, notice that in contrast to previous work on the unstructured radio network model [14, 15, 20], we do not make the simplifying assumption of having several independent communication channels. In our model, there is only one communication channel, i.e., two messages received at the same time always collide.

Every node has a unique identifier, which does not need to be in the range $1, \ldots, n$. Particularly, the algorithm does not perform explicit operations on the node's IDs. Instead, the ID is merely required to let a receiver recognize whether or not two different messages were sent by the same sender. In some papers on wireless sensor networks, it is argued that sensor nodes do not feature any kind of unique identification (such as a MAC number, for instance). In such a case, each node can randomly choose an ID uniformly from the range $[1 \ldots n^3]$ upon waking up. The probability that two nodes in the system end up having the same ID is bounded by $P_{ambIDs} \leq \binom{n}{2}\frac{1}{n^3} \in O(\frac{1}{n})$.

We denote by $\mathcal{N}_i$ the set of neighbors of node i and define $\mathcal{N}_i^+ = \mathcal{N}_i \cup \{i\}$. Further, let $\mathcal{N}_i^2$ be the two hop neighborhood of node i, i.e., the set of all nodes within distance at most 2 from i. The *degree* $\delta_i = |\mathcal{N}_i^+|$ of a node is the number of its neighbors. Finally, the color assigned to node v is denoted by $color(v)$ and p_i is the sending probability of node i in a given time-slot. Throughout the paper, we will use the following well-known mathematical fact.

FACT 1. *For all values of n and t with $n \geq 1$ and $|t| \leq n$, it holds that*

$$e^t\left(1-\frac{t^2}{n}\right) \leq \left(1+\frac{t}{n}\right)^n \leq e^t.$$

During the course of executing the algorithm, each node can be in various *node sets*. At any point in time, every node is in exactly *one* of the sets, i.e., the sets form a partition of the node set V. For future reference, we conclude the section with an enumeration of the different sets.

$\mathcal{Z}$:	set of sleeping nodes.
$\mathcal{W}$:	set of nodes in the waiting phase.
$\mathcal{L}$:	set of leaders.
$\mathcal{A}$:	set of nodes that are trying to become a leader.
$\mathcal{R}$:	set of slave nodes requesting a color-range from their leaders.
$\mathcal{K}$:	set of slave nodes that have received a color-range from their leader and are in the process of verifying a color from that range.
$\mathcal{C}$:	set of slave nodes that have decided on their color.

4. ALGORITHM

We start with an intuitive overview of Algorithm 1. The sequence of states that a node can be part of during the course of the algorithm is shown in Figure 1. Each solid arrow represents a state transition which is made when the event denoted by the arrow's label occurs. A dashed arrow between two states indicates the message type which is significant for the communication between nodes in these two states. However, note that the dashed arrows merely illustrate the purpose of the various message types in Algorithm 1, because *all* neighbors of a sending node – regardless of their current state – either receive the message or experience a collision. Furthermore, note that each node executes the algorithm after its wake-up without having any knowledge whether some of its neighbors have already started the algorithm beforehand.

The main idea of the algorithm is the following. The algorithm first elects a set of mutually independent *leaders* and associates each non-leader or *slave* with a leader within its neighborhood. This naturally leads to the notion of a *cluster* consisting of all nodes associated with the same leader. The task of the leader is to assign unique *color-ranges* to all nodes in its cluster. That is, within each cluster, no two nodes have the same color-range. Unfortunately, assigning an arbitrary color from these color ranges is not sufficient for a proper coloring since two neighboring nodes that are in different clusters may be assigned the same color-range. Hence, upon receiving a color-range from its leader, each node has to *verify* its color against other nodes from different clusters that may have received the same color-range.

The algorithmic difficulty of the above process stems from the fact that since nodes wake up asynchronously and do not have access to a global clock, the different phases of

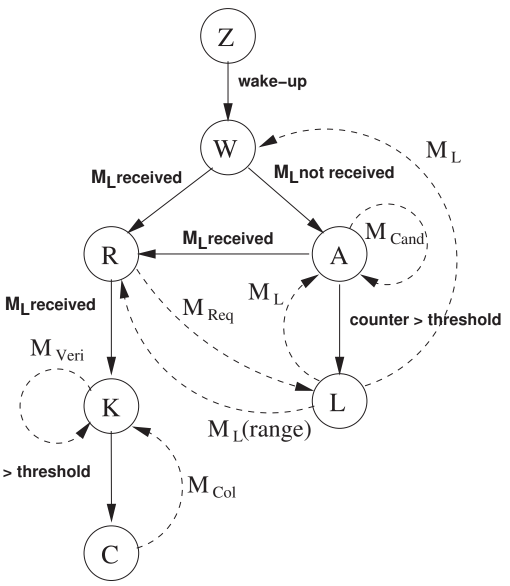

Figure 1: Sequence of states in Algorithm 1

the algorithm may be arbitrarily intertwined or shifted in time. Hence, while some nodes may still compete for becoming leader (not knowing that they are already covered by a leader), their neighbors may already be much more advanced in their coloring process.

A detailed code fragment of the procedure described above can be found in Algorithm 1. For the sake of clarity, we use the following notational shortcut in which a part of the program such as

```
A :    if counter ≥ $\lceil \sigma\Delta \ln n \rceil$ then
           state := $\mathcal{L}$; $q := q_L$; $j := 0$;
       end if
```

is meant to be executed only by nodes whose current state is $\mathcal{A}$. The same holds for all other states.

Upon waking up, each node first enters state $\mathcal{W}$, waiting for messages from neighboring leaders for $O(\Delta \log n)$ time-slots. If it does receive a message, it will directly join state $\mathcal{R}$. The reason for this initial waiting phase is that nodes waking up late in a region that is already covered by a leader should not try to compete for becoming a leader themselves. Nodes that have not received a message from a leader at the end of the waiting phase will then join set $\mathcal{A}$ and compete for leadership. The process of leader selection works as follows. In each time-slot, a competing node sends its ID with probability $q_s = 1/\Delta$. At the same time, starting from 0, it increases a counter in each time-slot.

As soon as a node $v \in \mathcal{A}$ receives a message M_L from a leader $v_\ell \in \mathcal{L}$, it stops competing, changes its state to $\mathcal{R}$, and sets v_ℓ as *its leader*. When receiving a message M_{Cand} from another node $v' \in \mathcal{A}$, v compares the counter value of v' to its own. If the two counters are within $\gamma\Delta \ln n$ of each other, v resets its own counter in Line 33. If on the other hand, a node in $\mathcal{A}$ succeeds in incrementing its counter up to the threshold of $\lceil \sigma\Delta \ln n \rceil$ (Line 6), it will become a leader, joining $\mathcal{L}$. Its color will be set to 0, and its sending probability increases to $q_L = 1/5$.

Algorithm 1 Coloring Algorithm (Node v)

```
upon wake-up do:
decided := $false$; sending := $false$;
$\kappa := 22$; $j := 0$; counter := 0; $q_L := 1/5$; $q_s := 1/\Delta$;
state := $\mathcal{W}$; mycolor := $-1$;      {Current state and color}
$q := q_s$;                         {Current sending probability}
$L_v := -1$;                        {Identity of $v$'s leader}
$\mathcal{Q} := \emptyset$;                        {Queue}
 1: for $\lceil \alpha\Delta \ln n \rceil$ time-slots do
 2:   if $M_L$ received then $L_v := m$.id; state := $R$;
 3: end for
 4: while $true$ do
 5:   counter = counter + 1;
 6:   $\mathcal{A}$: if counter $\geq \lceil \sigma\Delta \ln n \rceil$ then
 7:         state := $\mathcal{L}$; mycolor := 0; $q := q_L$; $j := 0$;
 8:      end if
 9:   $\mathcal{K}$: if counter $\geq \lceil \sigma\Delta \ln n \rceil$ then
10:         state := $\mathcal{C}$; decided := $true$;
11:      end if
12:   $\mathcal{L}$: if counter $\geq \lceil \beta \ln n \rceil$ then
13:         if $\mathcal{Q} \neq \emptyset$, remove tuple from $\mathcal{Q}$;
14:         counter := 0;
15:      end if
16:   sending := $\begin{cases} true & \text{with probability } q \\ false & \text{with probability } 1-q \end{cases}$
17:   if sending then
18:     $\mathcal{A}$: counter := max {counter, $\gamma\Delta \ln n + 1$};
19:         send $M_{Cand}$(id, counter);
20:     $\mathcal{R}$: send $M_{Req}$(id, lid);
21:     $\mathcal{K}$: counter := max {counter, $\gamma\Delta \ln n + 1$};
22:         send $M_{Veri}$(id, mycolor, counter);
23:     $\mathcal{C}$: send $M_{Col}$(id, mycolor);
24:     $\mathcal{L}$: if tuple (sid, $i$) is current tuple
25:         then send $M_L$(id, sid, $[i\kappa, \ldots, (i+1)\kappa - 1]$);
26:         else send $M_L$(id);
27:         end if
28:   else if not sending and received a message $m$ then
29:     $\mathcal{A}$: if $m = M_L$ then
30:           state := $\mathcal{R}$; counter := 0; $L_v := m$.id;
31:         else if $m = M_{Cand}$ then
32:           $\Delta c := |$counter $- m$.counter$|$;
33:           if $\Delta c \leq \lceil \gamma\Delta \ln n \rceil$ then counter := 0;
34:         end if
35:     $\mathcal{R}$: if $m = M_L \wedge m$.id $= L_v \wedge m$.sid = id then
36:           mycolor := $i\kappa$; state := $\mathcal{K}$;
37:         end if
38:     $\mathcal{K}$: if $m = M_{Col} \wedge m$.color = mycolor then
39:           counter := 0; mycolor := mycolor + 1;
40:         else if $m = M_{Veri}$ then
41:           $\Delta c := |$counter $- m$.counter$|$;
42:           if $\Delta c \leq \lceil \gamma\Delta \ln n \rceil$ then counter := 0;
43:         end if
44:     $\mathcal{L}$: if $m = M_{Req} \wedge m$.lid = id then
45:           add tuple ($m$.lid, $j$) to $\mathcal{Q}$; $j := j + 1$;
46:         end if
47:   end if
48: end while
```

The technique of using counters and critical ranges guarantees that quick progress is made in all parts of the network. Specifically, this method ensures that after a limited (constant) number of trials, at least one leader will emerge in *every region* of the graph. At the same time, we will prove that the method also guarantees with high probability that no two neighboring nodes become leader, i.e., the resulting set $\mathcal{L}$ is independent.

Slave nodes continue their algorithm by sending requests M_{Req} for a color-range to their respective leader. Upon receiving a message M_{Req}, a leader node assigns a *color-range* $[\phi, \ldots, \phi + \kappa - 1]$ consisting of κ consecutive colors to such a slave node by sending $M_L(range)$ messages. The constant $\kappa \leq 22$ denotes the number independent nodes than can maximally be packed in a disk of radius 2. For each subsequent slave, the leader shifts the color range such that within a single cluster, no two assigned color ranges overlap (The shifting is done in Line 45). Eventually, the slave node will receive one of these $M_L(range)$ messages from the leader, upon which it will take this color-range and move on to the verification state $\mathcal{K}$. Within a cluster, color-ranges do not interfere, but clusters assigned to different leaders may overlap and the same color-range may be assigned to neighboring nodes by different leaders. Therefore, a slave node cannot simply choose a color from its color-range, but it has to *verify* it. This verification procedure works similar as the initial leader selection procedure. All nodes assigned to the same color-range $[\phi, \ldots, \phi + \kappa - 1]$ begin by trying to get the first color ϕ. In the verification procedure, each node $v \in \mathcal{K}$ uses a counter value that is incremented until either v receives a message M_{Col} from a node $v' \in C$ that has decided on color ϕ; or v's counter reaches a certain threshold upon which v decides to take color ϕ and joins the set C. If v does not decide on ϕ, it continues the algorithm by verifying color $\phi + 1$ instead (Line 39). If in its next verification attempt, v does not decide on color $\phi + 1$ either (because a neighboring node decides on $\phi + 1$ earlier), v continues the same process with color $\phi + 2$, and so forth. In Section 5, we show that every node is capable of obtaining a color from the color-range assigned by its leader fast enough to avoid a deterioration of the algorithm's runtime. Once v decides on a color, it sends its color in a message M_{Col}, informing its neighbors that the particular color has already been taken.

Finally, the part of the leader is quite simple. Upon receiving a request message M_{Req} from one of its slaves, it sends for $\beta\Delta \ln n$ time-slots with probability $q_L = 1/5$ a message $M_L(range)$ with the assigned color-range. If it receives several requests within a short amount of time, a leader buffers these requests in an internal queue $\mathcal{Q}$. This is necessary in order to keep all messages within the size of $O(\log n)$ bits, assuming the ID space to be polynomial in n.

The four parameters in Algorithm 1, α, β, γ, and σ can be chosen as to trade-off the running time and the probability of correctness. Generally, the higher these parameters, the less likely the algorithm fails in producing a correct coloring. In order to obtain the high probability result of Section 5, the parameters are set as $\alpha \geq 2\gamma$, $\beta \geq \gamma$, and

$$\gamma = \frac{10e\kappa}{1 - \frac{1}{\Delta}} \left(\frac{5}{4}\right)^5, \quad \sigma \geq \left(\frac{5e}{4}\right)^\kappa \frac{2\kappa}{(1 - \frac{1}{\Delta})} + \gamma,$$

for $\Delta \geq 2$. Clearly, these constants are large. However, this is an artifact of our worst-case analysis. Our simulation results show that for most practical purposes, it is sufficient to set the constants to small values. In fact, the constants are small enough to yield a practically efficient coloring algorithm for wireless ad hoc and sensor networks that can be employed for the purpose of initializing the network.

5. ANALYSIS

In this section, we will prove that our algorithm is both correct and complete. *Correctness* means that no two adjacent nodes have the same color, *completeness* leaves no node without a color. Roughly, we begin by proving that the set of leaders $\mathcal{L}$ forms an independent set with high probability. Based on this result, it follows that all other color classes form correct independent sets as well, thus yielding a correct coloring. Furthermore, we prove that every node decides on a color after at most $O(\Delta \log n)$ time-slots. For clarity of exposition, we will omit the ceiling signs in our analysis. Further, let $c_i(t)$ be the value of the counter of node i at time t. We call a node *covered* if either itself or one of its neighbors is in $\mathcal{L}$. For succinctness, we define the constant $\lambda := (1-q_L)^5 = (4/5)^5$.

We start the analysis with a simple geometric argument bounding the number of independent nodes in the 2-hop neighborhood of a node to a constant factor κ and the number of two-hop neighbors to $\kappa\Delta$. Circles of radius 1/2 around two independent nodes cannot overlap. Let $\mathcal{L}_v^2$ be the number of leader nodes in the two hop neighborhood of node v. By a simple area argument, we can derive the following claim.

LEMMA 5.1. *Let $G=(V,E)$ be a unit disk graph. For every node $v \in G$, $|\mathcal{N}_v^2| \leq \kappa\Delta$ and $|\mathcal{L}_v^2| \leq \kappa$, where $\kappa \leq 22$.*

Another UDG property that we are going to use throughout the paper is that a node $v \in V$ can have at most 5 independent neighboring nodes [17].

We now state two lemmas that give us probabilistic bounds on the amount of time required until a message is correctly transmitted from a sender v to a receiver u in Algorithm 1. Notice that both lemmas are based on the assumption that the set $\mathcal{L}$ of leaders does indeed form a correct independent set.

LEMMA 5.2. *Assume $\mathcal{L}$ forms an independent set. Consider nodes u, u', and v with $(u,v),(u',v) \in E$, such that, $u \in V \setminus \mathcal{L}$ and $u' \in \mathcal{L}$. Let I and I' be time intervals of length $\gamma\Delta \ln n$ and $\gamma \ln n$, respectively. The probability P_{no} (P'_{no}) that v does not get a message from u (u') during an interval I (I') is bounded by $P_{no} \leq 1/n^{2\kappa}$.*

PROOF. The probability that a given node u succeeds in sending a message to v in an arbitrary time-slot is

$$\begin{aligned} P_s &= q\prod_{i\in\mathcal{N}_v}(1-p_i) = q\prod_{i\in\mathcal{N}_v\cap\mathcal{L}}(1-p_i)\prod_{j\in\mathcal{N}_v\cap(V\setminus\mathcal{L})}(1-p_j) \\ &\geq q\cdot(1-q_L)^{|\mathcal{N}_v\cap\mathcal{L}|}\cdot(1-q_s)^{|\mathcal{N}_v\cap(V\setminus\mathcal{L})|} \\ &\geq q\left(\frac{4}{5}\right)^5\left(1-\frac{1}{\Delta}\right)^{\Delta} \geq \frac{q\lambda}{e}\left(1-\frac{1}{\Delta}\right), \end{aligned}$$

where the last inequality follows from Fact 1. For the case $u \notin \mathcal{L}$, we can compute the probability P_{no} as

$$\begin{aligned} P_{no} &= (1-P_s)^{|I|} = \left(1-\frac{q_s\lambda}{e}\left(1-\frac{1}{\Delta}\right)\right)^{\gamma\Delta\ln n} \\ &= \left(1-\frac{\frac{\lambda}{e}\left(1-\frac{1}{\Delta}\right)}{\Delta}\right)^{\gamma\Delta\ln n} \leq n^{-\frac{\lambda}{e}(1-\frac{1}{\Delta})\gamma} < n^{-2\kappa}, \end{aligned}$$

where the last inequality follows from the definition of λ. An almost identical calculation proves the second claim of the lemma, i.e., the case $u \in \mathcal{L}$ with an interval of length $\gamma \ln n$.

$$\begin{aligned} P'_{no} &= (1-P_s)^{|I'|} = \left(1-\frac{q_L\lambda}{e}\left(1-\frac{1}{\Delta}\right)\right)^{\gamma\ln n} \\ &= \left(1-\frac{\lambda\log n}{5e\log n}\left(1-\frac{1}{\Delta}\right)\right)^{\gamma\ln n} \\ &\leq n^{-\frac{\lambda}{5e}(1-\frac{1}{\Delta})\gamma} < n^{-2\kappa}. \end{aligned}$$

□

For the next lemma, we first define the notion of a *successful* sending. A node v sends *successfully* in a given time-slot t if all nodes $v \in \mathcal{N}_v$ within the transmission range of v receive the message, i.e., there is no collision. The second helper lemma considers an arbitrary active node $v \in \mathcal{A}$ in the network graph. We show that with high probability, at least one node in v's neighborhood can send *successfully* during any interval of length $\eta\Delta \ln n$, where for convenience, η is defined as $\eta := \frac{2\kappa e^{\kappa}}{(1-q_L)^{\kappa}(1-\frac{1}{\Delta})}$.

LEMMA 5.3. *Assume $\mathcal{L}$ forms an independent set. Let $v \in \mathcal{A}$ be a node in the graph. Further, let I be a time interval of length $|I| = \eta\Delta\ln n$. With probability at least $1-\frac{1}{n^{2\kappa}}$, there is a time-slot $t \in I$ such that, at least one node $u \in \mathcal{N}_v^+ \cap \mathcal{A}$ sends* successfully.

PROOF. Consider a node $v \in \mathcal{A}$. By Lemma 5.1, there are at most $\kappa\Delta$ nodes in the 2-neighborhood of v, $|\mathcal{N}_v^+| \leq \kappa\Delta$. If in a certain time-slot t, a node $w \in \mathcal{N}_v^+ \cap \mathcal{A}$ manages to be the only sending node in $\mathcal{N}_v^2$, w sends successfully. Define $\lambda_2 := (1-q_L)^{\kappa}$ and let P_s be the probability that a successful sending occurs at a given time-slot. P_s is lower bounded by

$$\begin{aligned} P_s &= \sum_{w\in\mathcal{N}_v^+\cap\mathcal{A}}\left(p_w\prod_{\substack{u\in\mathcal{N}_v^2\\ u\neq w}}(1-p_u)\right) \\ &\geq \sum_{w\in\mathcal{N}_v^+\cap\mathcal{A}} p_w\prod_{u\in\mathcal{N}_v^2\setminus\mathcal{L}}(1-p_u)\prod_{u\in\mathcal{N}_v^2\cap\mathcal{L}}(1-p_u) \\ &\geq \sum_{w\in\mathcal{N}_v^+\cap\mathcal{A}} p_w\,(1-q_s)^{|\mathcal{N}_v^2\setminus\mathcal{L}|}(1-q_L)^{|\mathcal{N}_v^2\cap\mathcal{L}|} \\ &\underset{\text{Lm 5.1}}{\geq} q_s\,(1-q_s)^{\kappa\Delta}\,(1-q_L)^{\kappa} \\ &\underset{\text{Fact 1}}{\geq} \frac{e^{-\kappa}}{\Delta}\left(1-\frac{1}{\Delta}\right)\lambda_2 \geq \frac{e^{-\kappa}}{\Delta}\left(1-\frac{1}{\Delta}\right)\lambda_2 \end{aligned}$$

because $|\mathcal{N}_v^+\cap\mathcal{A}|$ for $v \in \mathcal{A}$ is at least 1. Finally, we compute the probability P_{no} that no node in $\mathcal{N}_v^+\cap\mathcal{A}$, manages to send successfully within the interval I as

$$\begin{aligned} P_{no} &= (1-P_s)^{|I|} \leq \left(1-\lambda_2\frac{e^{-\kappa}}{\Delta}\left(1-\frac{1}{\Delta}\right)\right)^{\eta\Delta\ln n} \\ &\underset{\text{Fact 1}}{\leq} e^{-\lambda_2 e^{-\kappa}(1-\frac{1}{\Delta})\eta\ln n} = n^{-\lambda_2 e^{-\kappa}(1-\frac{1}{\Delta})\eta} < \frac{1}{n^{2\kappa}}. \end{aligned}$$

The last step follows from the definitions of λ'_2 and η. □

The above lemma and its proof can be identically stated for nodes in set $\mathcal{K}$, instead of $\mathcal{A}$.

LEMMA 5.4. *Assume $\mathcal{L}$ forms an independent set. Let $v \in \mathcal{K}$ be a node in the graph. Further, let I be a time interval of length $|I| = \eta\Delta \ln n$. With probability at least $1 - \frac{1}{n^{2\kappa}}$, there is a time-slot $t \in I$ such that, at least one node $u \in \mathcal{N}_v^+ \cap \mathcal{K}$ sends* successfully.

Lemmas 5.2, 5.3, and 5.4 are based on the assumption that the set of leaders $\mathcal{L}$ forms an independent set. Therefore, in order to make full use of these lemmas, we need to prove that this assumption holds with high probability for the entire duration of the algorithm.

Intuitively, the reason for our claiming that the set of leaders $\mathcal{L}$ forms an independent set at all times is the following. Whenever a node v can send successfully, all neighboring nodes having a counter value within the critical range $\gamma\Delta \ln n$ of w's counter will reset their counter. Afterwards, none of these nodes can block v from becoming a leader anymore, because their counter is outside the critical range. The idea is that once a node joins $\mathcal{L}$, it has enough time to inform all neighbors of its being a leader (by means of a message M_L) before, potentially, a neighboring node may also join $\mathcal{L}$.

LEMMA 5.5. *The set of leaders $\mathcal{L}$ forms an independent set at all times t with probability of at least $1 - O(n^{-1})$.*

PROOF. At the beginning, when the first node wakes up, the claim certainly holds, because $\mathcal{L} = \emptyset$. We will now show that with high probability the claim will not be violated. Consider an arbitrary node $v \in \mathcal{A}$ and assume for contradiction that v is the *first node* to violate the independence of $\mathcal{L}$. We show that the probability of this event is at most n^{-2}. Consequently, the probability that there is at least one node $v \in V$ that violates the independence of $\mathcal{L}$ is bounded by $n \cdot n^{-2} = n^{-1}$.

The crucial observation is that since v is the *first* node to violate the independence of $\mathcal{L}$, we know that before the violation occurs, $\mathcal{L}$ is a correct independent set. Hence, under the assumption that v is the first node to create a violation, we can safely use Lemmas 5.2 and 5.3 until v becomes a leader and joins $\mathcal{L}$.

We first show that a node waking up in an already covered location will with high probability not become a leader. In Lines 1-3 of the algorithm, each node v waits for $\alpha\Delta \ln n$ time-slots after wake-up. By Lemma 5.2, if there exists a leader in v's neighborhood, it will be able to send a message to v during that interval without collision with probability exceeding $1 - \frac{1}{n^{2\kappa}}$. Throughout the proof, it is important to observe that the constant σ is defined to be large enough such that, $\sigma > \eta + \gamma$.

The more interesting case is when v does not receive a message from a leader during the waiting period, i.e., when it wakes up in an uncovered region. Let t_v be the time v reaches Line 4 of the algorithm, i.e., the first time it may send a M_{Cand} message, competing for leadership. We know by Lemma 5.3 that at least one node $w \in \mathcal{N}_v^+ \cap \mathcal{A}$ is able to send successfully during the interval $I = [t_v, t_v + \eta\Delta \ln n]$ with probability $1 - \frac{1}{n^{2\kappa}}$. Say this happens at time t_w^*. We distinguish the two cases $w = v$ and $w \neq v$, starting with the case $w = v$. According to Line 33 in the algorithm, all nodes $u \in \mathcal{N}_w^+ \cap \mathcal{A}$ whose counter $c_u(t_w^*)$ at time t_w^* is in the range

$$[c_w(t_w^*) - \gamma\Delta \ln n, \ldots, c_w(t_w^*) + \gamma\Delta \ln n]$$

will reset their counter at time t_w^*. In Line 18 of the algorithm, the sending node sets its counter to at least $\gamma\Delta \ln n + 1$. Therefore it holds that for all nodes $u \in \mathcal{N}_w \cap \mathcal{A}$,

$$|c_u(t_w^* + 1) - c_w(t_w^* + 1)| \; > \; \gamma\Delta \ln n. \qquad (1)$$

Notice that because $c_w(t_w^*) \geq \gamma\Delta \ln n + 1$ and the fact that new nodes joining $\mathcal{A}$ start with counter equal to 0, this also holds for new nodes $u \in \mathcal{A}$. Further, due to $\alpha > 2\gamma$, the above claim also holds for nodes that were still sleeping at the time t_w^* and started the waiting phase thereafter.

This means that *no node* $x \in \mathcal{A}$ adjacent to w has a counter close enough to c_w and therefore, no node $x \in \mathcal{A}$ is able to *stop* w from joining $\mathcal{L}$ after t_w^*. Hence, there remains only one way to prevent w from incessantly increasing its counter until reaching the threshold $\sigma\Delta \ln n$. Namely, the only way w can be prevented from joining $\mathcal{L}$ is if it receives a message M_L *before* its counter reaches the required threshold for joining $\mathcal{L}$. This message M_L must come from a node x that managed to join $\mathcal{L}$ before w. Specifically, as w had been able to send successfully at time t_w^*, we know that at time t_w^*, x's counter value was at least

$$c_x(t_w^*) \; > \; \gamma\Delta \ln n + c_w(t_w^*).$$

In other words, x must have joined $\mathcal{L}$ at least $\gamma\Delta \ln n$ time-slots *earlier* than w. By Lemma 5.2, the probability that w hears $x \in \mathcal{L}$ during one of these time-slots is at least $1 - \frac{1}{n^{2\kappa}}$. This result suffices because upon receiving M_L, $w = v$ will become a slave and hence, will not violate the lemma. In other words, the case $v = w$ can occur at most once for a node v.

In the case $v \neq w$, our argument becomes more tricky because potentially, a node v can reset its counter an infinite number of times and we must therefore handle the high probability results with care. In particular, we must be able to *bound the number of times* a node v can be in a situation that holds with high probability.

Again, let v be the node that, by assumption, is the first to violate the independence of $\mathcal{L}$. Let $w \in \mathcal{A}$ be the node that managed to send successfully at time t_w^* in the interval $[t_v, t_v + \eta\Delta \ln n]$. Recall that by Lemma 5.3, such a time-slot t_w^* exists with probability $1 - \frac{1}{n^{2\kappa}}$. After t_w^*, v will reset its counter. By the same "counter argument" (Equation (1)) as before, there remain the following two possibilities. Either w's counter reaches $\sigma\Delta \ln n$, i.e., w will join $\mathcal{L}$, or there exists a node $x \in \mathcal{N}_w$ that joins $\mathcal{L}$ earlier than w and w receives a message M_L from x. In the first case, w joins $\mathcal{L}$ at least $\gamma\Delta \ln n$ time-slots before v can join $\mathcal{L}$, too. Again, by Lemma 5.2, the probability that v receives a message M_L from $w \in \mathcal{L}$ is at least $1 - \frac{1}{n^{2\kappa}}$.

Finally, we have to consider the second possibility, namely that w itself is blocked from joining $\mathcal{L}$ by another leader x *after* time t_w^*. In this case v may not become covered within the next $\sigma\Delta \ln n$ time-slots. However, as node x must have become a leader in order to stop w, and because we assume v to be the first node violating independence, a constant fraction of the disk with radius 2 around v must have been covered by x (see Figure 2 for a visualization). Clearly, this scenario can happen only a *constant number of times* before v ends up being covered (possibly by itself). By Lemma 5.1, we can crudely bound this constant by κ. After v's resetting, the same arguments as above holds, i.e., there is at least one node $w \in \mathcal{N}_v^+ \cap \mathcal{A}$ capable of sending successfully during the interval $I = [t'_v, t'_v + \eta\Delta \ln n]$ and the entire reasoning

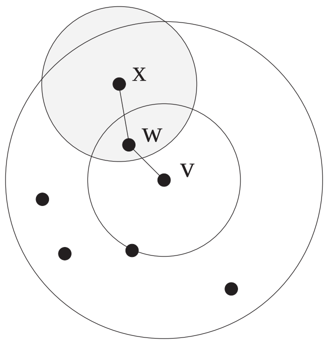

Figure 2: Whenever a node x joins $\mathcal{L}$ without covering v, a constant fraction of the large disk becomes covered.

repeats itself. By the argument in Figure 2, however, it can repeat itself at most κ times. In every repetition, there are two events that hold with high probability. First, one of v's neighbors can send successfully during a given interval and secondly, a node that joins $\mathcal{L}$ earlier than another has enough time to inform its neighbors. Both events hold with probability $1 - \frac{1}{n^{2\kappa}}$. Therefore, the probability P_v that v is the first node to violate independence is upper bounded by

$$P_v \;\leq\; \kappa \cdot \left(1 - \left(1 - \frac{1}{n^{2\kappa}}\right)\left(1 - \frac{1}{n^{2\kappa}}\right)\right) \;\leq\; 2\kappa n^{-2\kappa}, \quad (2)$$

which, together with $|V| = n$, concludes the proof. $\square$

Having the high probability result of Lemma 5.5 renders the remainder of the analysis easier since we can now use Lemmas 5.2, 5.3, and 5.4 without restriction. What remains to be shown is that no two neighboring nodes decide to take the same color, or in other words, that all color classes form correct independent sets at all times.

LEMMA 5.6. *With probability at least $1-O(n^{-1})$, all color classes form independent sets at all times throughout the execution of the algorithm.*

PROOF. In the algorithm, the leader assigns unique color-ranges of the form $[i, \ldots, i + \kappa - 1]$ to all nodes within its cluster. The idea is that following the geometric argument of Lemma 5.1, at most κ neighboring nodes (each of which is assigned to a different leader node) can be assigned the same color-range. Therefore, there are enough colors for each of the at most κ nodes to get one unique color from its assigned range. All nodes assigned to the same color-range $[i, \ldots, i + \kappa - 1]$ will first try to *verify* color i, by starting incrementing their counter upon sending their first M_{Veri} message. The whole process is then a copy of the initial leader election mechanism discussed above. Nodes increase their counter trying to reach the threshold $\sigma\Delta \ln n$ upon which they become eligible to definitely decide on color i. Once a node decides on a color, it starts sending messages M_{Col} messages in order to inform neighboring nodes. There is only one difference to the initial leader election mechanism. That is, if a node v that tries to obtain color i receives a M_{Col} (corresponding to M_L) from a node $u \in C_i$, v remains in state $\mathcal{K}$, but it starts attempting to verify the next higher color, $i + 1$.

By exactly the same argument as in the proof of Lemma 5.5 (replacing Lemma 5.3 by Lemma 5.4), it is clear that a single phase (i.e., the verification of a color i) of the above process succeeds with probability $1 - O(n^{-1})$. If everything goes right, all nodes will have decided on a unique color in their color-range after at most κ phases. The probability P_{no} that at least one of these κ phases is not successful is therefore $P_{no} \in O(n^{-1})$. $\square$

For practical purposes, the assignment of colors to nodes plays a crucial role. Generally, the colors assigned to each node should be as "low" as possible. If the vertex coloring in the graph is used for setting up a *time-division scheduling* in a wireless network, for instance, the bandwidth assigned to a node v is often inversely proportional to the value of the *highest color* in its neighborhood. The highest color assigned to a neighbor of a node v by Algorithm 1 is dependent only on *local graph properties*. This allows nodes located in low density areas of the network to send more frequently than nodes in dense and congested parts.

THEOREM 5.7. *Let $\theta_v := \max_{w \in N_v^2} \delta_w$ be the maximum node degree in N_v^2 and let χ_v be the highest color assigned to a node in N_v^+. With high probability, Algorithm 1 produces a coloring such that, for all $v \in V$, $\chi_v \leq \kappa \cdot \theta_v \in O(\theta_v)$.*

PROOF. Let $w \in \mathcal{L}$ be a leader and let s_w be the number of slaves assigned to w. Each leader $w \in \mathcal{L}$ assigns unique color ranges $[i\kappa, \ldots, i\kappa + \kappa - 1]$, for $i = 0, \ldots, s_w - 1$ to its slaves. Hence, for each slave v assigned to leader w, it holds, $\mathit{color}(v) \leq s_w\kappa + \kappa - 1$ with high probability. Since $s_w \leq \delta_w$ and every node $u \in N_v$ is assigned to a leader $w \in N_v^2$, the theorem follows. $\square$

Correctness of the algorithm now follows as an easy consequence of Lemmas 5.5 and 5.6, as well as Theorem 5.7.

THEOREM 5.8. *Algorithm 1 produces a correct coloring with at most $\kappa\Delta$ colors with probability $1 - O(n^{-1})$.*

It remains to prove the claimed running time of the algorithm. The following theorem shows that with high probability, each node decides within time $O(\Delta \log n)$ after its waking up.

THEOREM 5.9. *Every node decides on its color within time $O(\Delta \log n)$ after its wake-up with probability $1 - O(n^{-1})$.*

PROOF. Let $T^i_{\mathcal{Y}}$ be the number of time-slots a node v_i spends in set $\mathcal{Y}$, and let T^i be the total running time of node v_i, i.e., the number of time-slots from its wake-up to its final decision for a color. For each node v_i, we have

$$T^i \;=\; T^i_{\mathcal{W}} + T^i_{\mathcal{A}} + T^i_{\mathcal{R}} + T^i_{\mathcal{K}}$$

For leaders $v_i \in \mathcal{L}$ it holds $T^i_{\mathcal{R}} = T^i_{\mathcal{K}} = 0$, because leaders decide on their color – 0 – the moment they join $\mathcal{L}$. Also, note that for slaves, $T^i_{\mathcal{A}}$ can be 0 in case a node receives a message M_L from a leader while waiting in state $\mathcal{W}$. In the sequel, we show the running time for each part individually.

By definition of the algorithm, in Line 1, it is clear that

$$T^i_{\mathcal{W}} \;=\; \alpha\Delta \ln n. \quad (3)$$

As for $T^i_{\mathcal{A}}$, let t^*_v be the time-slot in which node v_i joins set $\mathcal{A}$. By Lemma 5.3, a node w in $N_v^+ \cap \mathcal{A}$ is able to send *successfully* in $[t^*_v \ldots, t^*_v + \eta\Delta \ln n]$, with probability

$1-O(n^{-2\kappa})$. Now, either w joins $\mathcal{L}$ $\sigma\Delta\ln n$ time-slots thereafter, or there exists a node $x \in N_w^+ \in N_v^2$ that becomes leader before w, that is, before time $t_x \leq t_v^* + \eta\Delta\ln n + \sigma\Delta\ln n$. In the first case, v is covered and changes to state $\mathcal{R}$ by Lemma 5.2 with probability $1 - O(n^{-2\kappa})$ (because v will receive a message M_L from w within that time). By Lemma 5.1 and the fact that with probability $1 - O(n^{-1})$, the set $\mathcal{L}$ is independent (Lemma 5.5), the second case can occur at most κ times. Therefore,

$$T_{\mathcal{A}}^i \leq \kappa(\eta\Delta\ln n + \sigma\Delta\ln n) = \kappa(\eta+\sigma)\Delta\ln n \qquad (4)$$

with probability $1 - O(n^{-1})$.

The time $T_{\mathcal{R}}^i$ is the time between v starting to request a color-range from its leader to the time leader $w \in \mathcal{L}$ succeeds in sending to v without collision. We divide $T_{\mathcal{R}}^i$ into two parts. First, by Lemma 5.2, v is able to send its request M_{Req} to w within time $\gamma\Delta\ln n$ with probability $1-O(n^{-2\kappa})$. Upon reception, w may queue v's request until it has served all its other slaves. By Line 12 of the algorithm, a leader tries for $\beta\ln n$ time-slots to send a message M_L to each of its slave nodes before moving on to the next request, if available. Because $\beta \geq \gamma$, Lemma 5.2 holds for w's response to v with high probability. Because w can have at most Δ slaves, $T_{\mathcal{R}}^i$ is at most

$$T_{\mathcal{R}}^i \leq \gamma\Delta\ln n + \Delta\cdot\beta\ln n = (\gamma+\beta)\Delta\ln n$$

Finally, $T_{\mathcal{K}}^i$ – the time it takes for a slave to verify its color, after having been assigned a color-range – follows a similar argument as $T_{\mathcal{A}}^i$. By Lemma 5.4, a node w in $N_v^+ \cap \mathcal{K}$ is able to send *successfully* in $[t_v^* \ldots, t_v^* + \eta\Delta\ln n]$ with probability $1 - O(n^{-2\kappa})$. Again, either w decides on its color $\sigma\Delta\ln n$ time-slots thereafter, or there exists a node $x \in N_w^+ \in N_v^2$ that decides on a color before w. By a geometric argument, there can be at most 2κ leaders in v's three hop neighborhood and therefore, at most as many nodes in N_v^2 that are assigned to the same color-range. So, with probability $1 - O(n^{-1})$, we have

$$T_{\mathcal{K}}^i \leq 2\kappa(\eta\Delta\ln n + \sigma\Delta\ln n) = 2\kappa(\eta+\sigma)\Delta\ln n. \qquad (5)$$

Plugging equations 3 to 5 together, we get

$$\begin{aligned} T^i &= T_{\mathcal{W}}^i + T_{\mathcal{A}}^i + T_{\mathcal{R}}^i + T_{\mathcal{K}}^i \\ &\leq (\alpha + 3\kappa(\eta+\sigma) + \gamma + \beta)\Delta\ln n \in O(\Delta\log n), \end{aligned}$$

which concludes the proof. □

6. CONCLUSIONS

Setting up an initial structure in newly deployed ad hoc and sensor networks is a challenging task that is of great practical importance. In this paper, we have given a randomized algorithm that computes an initial coloring from scratch. This is a step towards the ultimate goal of establishing an efficient medium access control scheme. In view of the trivial $\Omega(\Delta)$ lower bound for vertex coloring in our model, there remains a gap of $O(\log n)$ between the lower bound and our upper bound. Closing this chasm is an interesting open problem.

Another direction for future research is to address the issue that our algorithm is based on the assumption that nodes know an estimate of n and Δ. In single-hop radio networks with synchronous wake-up, there are efficient methods enabling nodes to approximately count the number of their neighbors [10]. If such techniques could be adapted to an asynchronous multi-hop scenario, nodes might be able to estimate the local maximum degree, which could then be used instead of Δ throughout the algorithm.

7. ACKNOWLEDGEMENTS

We are indebted to Susan Babutzka for many valuable comments and numerous fruitful discussions on the topic.

8. REFERENCES

[1] I. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci. Wireless Sensor Networks: A Survey. *Computer Networks Journal*, 38(4):393–422, 2002.

[2] R. Bar-Yehuda, O. Goldreich, and A. Itai. On the Time-Complexity of Broadcast in Radio Networks: an Exponential Gap between Determinism and Randomization. In *Proceedings 6th ACM Symposium on Principles of Distributed Computing (PODC)*, pages 98–108, 1987.

[3] R. Cole and U. Vishkin. Deterministic Coin Tossing with Applications to Optimal Parallel List Ranking. *Inf. Control*, 70(1):32–53, 1986.

[4] I. Finocchi, A. Panconesi, and R. Silvestri. Experimental Analysis of Simple, Distributed Vertex Coloring Algorithms. In *Proceedings of the 13th ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 606–615, 2002.

[5] L. Gasieniec, A. Pelc, and D. Peleg. The Wakeup Problem in Synchronous Broadcast Systems (Extended Abstract). In *Proceedings of the 19th ACM Symposium on Principles of Distributed Computing (PODC)*, pages 113–121, 2000.

[6] A. Goldberg, S. Plotkin, and G. Shannon. Parallel symmetry-breaking in sparse graphs. In *Proceedings of the 19th Annual ACM Conference on Theory of Computing (STOC)*, pages 315–324, 1987.

[7] A. V. Goldberg and S. A. Plotkin. Parallel $(\Delta+1)$-Coloring of Constant-degree Graphs. *Information Processing Letters*, 25:241–245, 1987.

[8] D. A. Grable and A. Panconesi. Fast Distributed Algorithms for Brooks-Vizing Colourings. In *Proceedings of the 9th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 473–480. Society for Industrial and Applied Mathematics, 1998.

[9] W. R. Heinzelman, A. Chandrakasan, and H. Balakrishnan. Energy-Efficient Communication Protocol for Wireless Microsensor Networks. In *Proceedings of the 33rd Hawaii International Conference on System Sciences (HICSS)*, page 8020. IEEE Computer Society, 2000.

[10] T. Jurdzinski, M. Kutylowski, and J. Zatopianski. Energy-Efficient Size Approximation of Radio Networks with No Collision Detection. In *Proceedings of the 8th Annual International Conference on Computing and Combinatorics (COCOON)*, pages 279–289, 2002.

[11] T. Jurdzinski, M. Kutylowski, and J. Zatopianski. Weak Communication in Radio Networks. In *Proceedings of Euro-Par*, pages 397–408, 2002.

[12] T. Jurdzinski and G. Stachowiak. Probabilistic Algorithms for the Wakeup Problem in Single-Hop Radio Networks. In *Proceedings of 13th Annual International Symposium on Algorithms and Computation (ISAAC)*, pages 535–549, 2002.

[13] S. O. Krumke, M. V. Marathe, and S. S. Ravi. Models and Approximation Algorithms for Channel Assignment in Radio Networks. *Wireless Networks*, 7(6):575–584, 2001.

[14] F. Kuhn, T. Moscibroda, and R. Wattenhofer. Radio Network Clustering from Scratch. In *Proceedings of 12th Annual European Symposium on Algorithms (ESA)*, pages 460–472.

[15] F. Kuhn, T. Moscibroda, and R. Wattenhofer. Initializing Newly Deployed Ad Hoc and Sensor Networks. In *Proceedings of 10th Annual International Conference on Mobile Computing and Networking (MOBICOM)*, pages 260–274, 2004.

[16] N. Linial. Locality in Distributed Graph Algorithms. *SIAM Journal on Computing*, 21(1):193–201, 1992.

[17] M. V. Marathe, H. Breu, H. B. Hunt III, S. S. Ravi, and D. J. Rosenkrantz. Simple Heuristics for Unit Disk Graphs. *Networks*, 25:59–68, 1995.

[18] G. D. Marco and A. Pelc. Fast Distributed Graph Coloring with $O(\Delta)$ Colors. In *Proceedings of the 12th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 630–635, 2001.

[19] M. J. McGlynn and S. A. Borbash. Birthday Protocols for Low Energy Deployment and Flexible Neighbor Discovery in Ad Hoc Wireless Networks. In *Proceedings of the 2nd ACM International Symposium on Mobile Ad Hoc Networking and Computing (MOBIHOC)*, pages 137–145. ACM Press, 2001.

[20] T. Moscibroda and R. Wattenhofer. Efficient Computation of Maximal Independent Sets in Unstructured Multi-Hop Radio Networks. In *Proceedings of 1st IEEE International Conference on Mobile Ad-Hoc and Sensor Systems (MASS)*, 2004.

[21] T. Moscibroda and R. Wattenhofer. Maximal Independent Sets in Radio Networks. In *Proceedings of the 23rd ACM Symp. on Principles of Distributed Computing (PODC)*, 2005.

[22] K. Nakano and S. Olariu. Energy-Efficient Initialization Protocols for Single-Hop Radio Networks with No Collision Detection. *IEEE Trans. Parallel Distributed Systems*, 11(8):851–863, 2000.

[23] K. Nakano and S. Olariu. Randomized Initialization Protocols for Radio Networks. pages 195–218, 2002.

[24] A. Panconesi and R. Rizzi. Some Simple Distributed Algorithms for Sparse Networks. *Distributed Computing*, 14(2):97–100, 2001.

[25] D. Peleg. *Distributed Computing: A Locality-Sensitive Approach.* SIAM Monographs on Discrete Mathematics and Applications, 2000.

[26] S. Ramanathan and E. L. Lloyd. Scheduling Algorithms for Multi-Hop Radio Networks. In *Conference proceedings on Communications architectures & protocols (SIGCOMM)*, pages 211–222. ACM Press, 1992.

[27] F. A. Tobagi and L. Kleinrock. Packet Switching in Radio Channels: Part II - The Hidden Terminal Problem in Carrier Sense Multiple Access and the Busy Tone Solution. COM-23(12):1417–1433, 1975.

[28] A. Woo and D. E. Culler. A Transmission Control Scheme for Media Access in Sensor Networks. In *Proceedings of the 7th International Conference on Mobile Computing and Networking (MOBICOM)*, pages 221–235. ACM Press, 2001.