Fast Beeping Protocols for Deterministic MIS and $(\Delta + 1)$-Coloring in Sparse Graphs (Extended Version)

Joffroy Beauquier*[‡], Janna Burman*[‡], Fabien Dufoulon*[‡] and Shay Kutten[†][‡]
* LRI, Université Paris-Sud, CNRS, Université Paris-Saclay, France. beauquier@lri.fr, burman@lri.fr, dufoulon@lri.fr
[†] Technion - Israel Institute of Technology, Haifa, Israel. kutten@ie.technion.ac.il

***Abstract*—The beeping model is an extremely restrictive broadcast communication model that relies only on carrier sensing. We consider two problems in this model: $(\Delta{+}1)$-vertex coloring and maximal independent set (MIS), for a network of unknown size n and unknown maximum degree Δ. Solving these problems allows to overcome communication interferences, and to break symmetry, a core component of many distributed protocols. The presented results apply to general graphs, but are efficient in graphs with low edge density (sparse graphs), such as bounded degree graphs, planar graphs and graphs of bounded arboricity. We present $O(\Delta^2 \log n + \Delta^3)$ time deterministic uniform MIS and coloring protocols, which are asymptotically time optimal for bounded degree graphs. Furthermore, we devise $O(a^2 \log^2 n + a^3 \log n)$ time MIS and coloring protocols, as well as $O(a^2\Delta^2 \log^2 n + a^3\Delta^3 \log n)$ time 2-hop MIS and 2-hop coloring protocols, where a is the arboricity of the communication graph. Building upon the 2-hop coloring protocols, we show how the strong CONGEST model can be simulated and by using this simulation we obtain an $O(a)$-coloring protocol. No results about coloring with less than $\Delta + 1$ colors were known up to now in the beeping model.**

I. Introduction

The discrete beeping model was introduced by Cornejo and Kuhn [1] to provide a convenient formal framework for studying radio networks having severe restrictions on communication capabilities, yet where widely applicable protocols can be designed and analytically proven in an efficient manner. Since protocol executions in distributed computing are frequently hard to grasp (even with simulations or experiments), having formal models is crucial for both practical and theoretical reasons. In the discrete beeping model, time is divided into synchronous rounds, and in each round, a node can either listen or transmit a unary signal (beep) to all its neighbors. The possibility to directly transmit a beep to a node is defined by a *static communication graph*, and nodes have absolutely no knowledge of this graph. A beeping node receives no feedback, while a silent one can only detect that either at least one of its neighbors beeped or that all of them were silent. A listening node does not receive the identifiers of its beeping neighbors, as a beep is merely a detectable burst of energy. Protocols can use the synchronous nature of the rounds to transmit information through beeps, but doing so impacts the time complexity in a quantifiable manner. This work studies how this difficulty can be overcome.

Applications of this model range from radio networks with reduced network stacks [1], such as energy-limited sensor networks, which can provide improved speed, low cost and less transmission errors, to biological networks [2], where the beeping model allows to study the efficiency of natural protocols. Indeed, most biological systems communicate in a primitive manner. Fireflies communicate through flashes of light [3] and cells through the diffusion of specific chemical markers [4].

Different applications will result in different communication graphs. Graphs with low edge density are said to be *sparse*. The *maximum degree* and the *arboricity* of a graph are measures of its edge density, where low values indicate sparse graphs. Contrarily to graphs with low maximum degree, low arboricity graphs can be seen as graphs which are "globally" sparse but may be "locally" dense. Many real-world networks are sparse [5]. In particular, graphs embedded in some surface, for example the plane, have low arboricity.

The distributed vertex coloring and maximal independent set (MIS) problems are fundamental building blocks in protocol design. The coloring problem consists in assigning colors to nodes such that no two neighboring nodes (sharing an edge in the communication graph) have the same color. The MIS problem consists in choosing a set of nodes in the communication graph such that no two nodes in the set are neighbors, and such that any node not in the set has a neighbor in that set. Solving these problems is important for dealing with the interferences inherent to the beeping model. More specifically, a coloring can be used to allocate resources that cannot be shared by neighboring nodes. Nodes in an MIS can act as cluster heads in order to coordinate actions, and participate in a network backbone construction.

Serving as important primitives for protocol design in the beeping model, MIS and coloring problems have received a lot of attention (see Sect. I-B). Efficient probabilistic solutions were proposed for general graphs. However, the more difficult deterministic case, useful whenever random behavior is inappropriate or deterministic guarantees are required, has received much less attention. In this work, we are interested in designing deterministic protocols having efficient time complexity.

[‡] This work has been supported by the Israeli-French Maimonide research project grant and by an Israeli Ministry of Science and Technology grant.

A. Preliminaries

Let $[k]$ be the set $\{1, \ldots, k\}$. For any two integers a, b ($\in \mathbb{Z}$) and any positive integer k ($\in \mathbb{N}^{>0}$), let $a \equiv b \mod k$ denote the *congruence relationship* between a and b such that $a \mod k = b \mod k$. The operator $\|$ is used for the *string concatenation*. For any positive integer k, $l(k)$ is the *length* of the binary representation of k, i.e., $l(k) = 1 + \lfloor \log_2 k \rfloor$. For any function $f : \mathbb{N}^{>0} \mapsto \mathbb{N}^{>0}$ and any positive integer k, $f_i(k)$, where $i \in [l(k)]$, denotes the ith *most significant bit* of $f(k)$'s binary representation.

The *communication network* is represented by a simple connected undirected graph $G = (V, E)$, where V is the node set and E the edge set. The *network size* $|V|$ is also denoted by n, the *diameter* by D and the *maximum degree* by Δ. For a node $v \in V$, the *neighbors* of v are $\mathcal{N}(v) = \{u \in V \text{ s.t. } (u, v) \in E\}$ and its *degree* is $deg(v) = |\mathcal{N}(v)|$. Nodes have unique identifiers (ids). This property is essential in order to break symmetry in deterministic protocols. The *identifier* of a node $u \in V$, $id(u)$, is an integer from $[N]$ where $N = n^c$ with a constant $c > 1$. N is an upper bound on the total number of nodes in G. The *length* of $id(u)$ is denoted by $l(u)$. Then, the *maximum length* over all ids in G is $l_{max} = \max_{u \in V} l(u)$. We have $l_{max} = O(\log N) = O(\log n)$.
The *distance* between two nodes u and v in G is $dist(u, v)$. The *square graph* of G is the graph $G^2 = (V, E_s)$, where $E_s = \{(u, v) \mid u, v \in V, dist(u, v) \leq 2\}$. $G[R]$ denotes the subgraph of G *induced* by $R \subset V$. Its edges ($E_G[R]$) are the edges of G connecting two vertices in R. The *arboricity* of G, denoted by $a(G)$ or just a, is the minimum number of disjoint forests into which the edge set E can be partitioned. Equivalently, Nash-Williams [6] proved that arboricity is also a measure of density, i.e., $a = \max\limits_{R \subseteq V, |R| \geq 2} \frac{|E_G[R]|}{|R|-1}$.

B. Related Work

In [7], round complexity lower bounds are given for the MIS and $(\Delta + 1)$-coloring problems. These bounds are $\Omega(\log n)$ and $\Omega(\Delta + \log n)$ respectively. They were obtained assuming randomized algorithms, and thus apply to both deterministic and randomized ones. In the latter case, the solution or the running time is guaranteed with high probability (w.h.p.). Moreover, these bounds apply to a stronger variant of the beeping model (with collision detection). In this variant, listening nodes can distinguish between a single beep and the superposition of multiple beeps (a collision).

In [1], the authors present the first coloring protocol for the beeping model. It outputs a correct coloring after $O(\Delta + \log n)$ rounds w.h.p. Following this paper, randomized MIS and coloring protocols were designed for the beeping model with collision detection, in a series of papers ([4], [8], [9]). These protocols achieve optimal round complexity, but assume collision detection. Moreover, the resulting colorings often employ more than $\Delta + 1$ colors. These protocols can be translated to the weaker beeping model (with no collision detection) with an $\Omega(\log n)$ multiplicative factor.

Schneider and Wattenhofer [10] solve *deterministic* MIS in radio networks with collision detection. Although the term "beeping model" does not appear in [10], the presented protocol straightforwardly works in this model. It is time optimal for growth-bounded graphs (GBG). These are graphs where, for any given node v and integer r, the number of nodes in any *independent set* (see definition in Sect. I-D) within distance r of v is bounded by $f(r)$, which is polynomial in r. However, this property does not cover all bounded degree graphs, trees, planar graphs, or more generally, sparse graphs.
The round complexities of different MIS and coloring protocols are compared below (see respectively Figure 1 and 2). The only deterministic protocols are those in [10] and in the present paper. Some protocols require K, an upper bound on Δ.

Fig. 1: MIS protocols

Ref	Time	Comments
[8]	$O(\log^2 n)$ w.h.p.	anonymous nodes
[10]	$O(\log n)$	GBG, deterministic
Here	$O(\Delta^2 \log n + \Delta^3)$	deterministic

Fig. 2: Coloring protocols

Ref	Time	Comments
[9]	$O(\Delta \log n + \log^2 n)$ (w.h.p.)	$\Delta + \log n$ colors
[9]	$O(K \log^2 n)$ (w.h.p.)	$K + 1$ colors
[1]	$O(\Delta + \log n)$ (w.h.p.)	$O(K)$ colors
Here	$O(\Delta^2 \log n + \Delta^3)$, deterministic	$\Delta + 1$ colors

C. Protocol-related Definitions

In the *beeping model*, an execution proceeds in synchronous rounds (there are synchronized local clocks and all nodes start at the same time: *synchronous start*). In each round, nodes synchronously execute the following steps:

1) Send: Each node beeps (instruction $BEEP$ in protocols) or listens ($LISTEN$ in protocols). Beeps are transmitted to all neighbors of the beeping node.
2) Receive: If a node beeped in the previous step, then it learns no information from its neighbors. Otherwise, it knows whether or not at least one of its neighbors beeped during the previous step of the same round.
3) Process: Each node performs local computations.

One of the most common message passing models is the CONGEST model of edge bandwidth B [11]. It is stronger than the beeping model, as nodes communicate by sending messages of maximum length B (commonly $O(\log n)$) in a round. Different messages can be sent to different neighbors and nodes receive the full content of all incoming messages.

We adopt the classical definitions. The *state* of a node is the vector of the values of its variables. A variable var of a node v is explicitly associated to v using a subscript var_v. A *configuration* is a vector of the states of all nodes. An *execution* proceeds in rounds and is defined by the sequence of the configurations at the end of each round, starting from an initial configuration. If the same configuration (resp. state of a node) is repeated indefinitely at the end of each round, we say that this configuration (resp. the state) is *terminal*. When such a configuration is reached, it is said that the system/protocol has *terminated*, or that *termination* has occurred. A *problem* is given as a first order predicate over configurations. A protocol is said to *solve a problem* if each execution terminates, and each terminal configuration satisfies the predicate of the problem specification. The *round complexity* (time complexity) of a protocol is the number of rounds needed to reach a terminal configuration in the worst case. A protocol is said to be *uniform* in a parameter p if it does not depend on the value of p. It is said to be *locally termination detecting*, or simply *locally terminating*, if for any given node v, v detects if it has reached a terminal state.

In the beeping model, protocols must specify what is done in each round. Due to the nature of the communication model, each action is performed on a sequence of consecutive rounds. For instance, a node may have to wait for a round of silence, or beep only every k rounds. At the code level, this type of action is expressed by a loop. As it will appear later, in some complex protocols, such loops are nested. For the sake of clarity, we will name the sequence of rounds in the innermost loop the $\mathcal{L}_1$-phase, the sequence of loops in the loop just above, the $\mathcal{L}_2$-phase, and so on.
We extend previous definitions concerning protocols to $\mathcal{L}_i$-phases, in particular uniformity and termination. We consider terminal $\mathcal{L}_i$-phase states (states that no longer change in this $\mathcal{L}_i$-phase), *locally terminating* $\mathcal{L}_i$-phases (any given node v detects when it has reached a terminal $\mathcal{L}_i$-phase state) and uniform $\mathcal{L}_i$-phases (when the range of the loop index is unknown). The problem of detecting when a given $\mathcal{L}_i$-phase has ended (terminated) *for all nodes* raises the question of synchronizing the start of the following $\mathcal{L}_i$-phase.
We solve this problem by using *$\mathcal{L}_i$-synchronization points*, represented by $\lightning_i$ in the code. Upon reaching an $\mathcal{L}_i$-synchronization point (after having reached a terminal $\mathcal{L}_i$-phase state), any given node v waits for all of its neighbors to reach a terminal $\mathcal{L}_i$-phase state before executing the following $\mathcal{L}_i$-phase, if there is any. $\mathcal{L}_i$-synchronization points require locally terminating $\mathcal{L}_i$-phases, so that any given node v can detect when all of its neighbors have reached the synchronization point. The method for detecting that was first introduced in Förster *et al.* [12], with the "Balanced Execution Technique" (BET). However, BET only guarantees $\mathcal{L}_1$-synchronization points. In appendix A, we extend BET to guarantee $\mathcal{L}_i$-synchronization points for any $i \geq 1$. The extension, referred to as EBET, is crucial in the design of complex uniform protocols in the beeping model.

We call a protocol a *competition protocol* when nodes are "eliminated" round after round until the "surviving" nodes form an independent set (possibly empty). In this paper, we only consider competition protocols where the elimination process is deterministic and depends on identifier comparison.

D. Problem Specifications

The predicates (over configurations) defining the problems considered in the paper can be naturally obtained from the definitions given below.
A set $I \subseteq V$ of vertices is said to be an *independent set* if for any u, v in I, u and v are not neighbors in G. An independent set I is *maximal* (MIS) if any vertex in $V \setminus I$ has a neighbor in I. A 2-hop MIS of G is an MIS of its square graph G^2.
A set $J \subseteq V$ of vertices is said to be a (t,s)*-ruling set* [13], if for any two vertices $u, v \in J$, $dist(u,v) \geq t$, and for any vertex $v \in V \backslash J$, there exists a vertex $u \in J$ such that $dist(u,v) \leq s$. With this definition, an MIS is a $(2,1)$-ruling set. A forest is said to be a (t,s)*-ruling forest* if the roots are a (t,s)-ruling set and the trees are of depth at most s.
A *c-coloring col* is a function from V into a set of colors $[c]$ such that $\forall (u,v) \in E \; col(u) \neq col(v)$. Notice that in a c-coloring, nodes with the same color constitute an independent set. It is thus possible to construct an MIS from them. A 2-hop coloring of G is a coloring of its square graph G^2.
Any given function $colorD$ is a *d-defective c-coloring* [14] if $\forall v \in V$, $colorD(v) \in [c]$ and v has at most d neighbors colored with $colorD(v)$. We say that $colorD$ has a *defect* of d. An edge where both endpoints have different colors is said to be a *non defective edge*, otherwise it is said to be a *defective edge*. With this definition, a (proper) coloring is a 0-defective coloring.

E. Contributions

The first contribution of this paper is a tool for analyzing competition protocols: the *labeled "deterministic"* competition graphs (Sect. II). Using such graphs in protocol analysis is inspired by [15]. Here we adapt this technique to the case of deterministic competition protocols. This general tool may be useful also in future studies of MIS and coloring.

The second and main contribution of the paper is a series of *uniform* MIS and coloring protocols. First, we present $O(\Delta^2 \log n + \Delta^3)$ deterministic uniform protocols for MIS and $(\Delta + 1)$-coloring, where Δ is the unknown maximum degree (Sect. III). These protocols are time optimal for *bounded degree* graphs. They also scale well to graphs with *polylogarithmic* Δ. Indeed, in these graphs, the time complexity is polylogarithmic in regards to n, which is very efficient.
Then, we extend the previous protocols with time complexity dependent on Δ, to protocols with time complexity dependent on the arboricity a (Sect. IV). We get $O(a^2 \log^2 n + a^3 \log n)$ time MIS and $(\Delta+1)$-coloring uniform protocols. This results in efficient polylogarithmic time complexity for the large family of graphs where $a = O(\log^c n)$. Finally, we extend the previous protocols into $O(a^2\Delta^2 \log^2 n + a^3\Delta^3 \log n)$ time 2-hop coloring and 2-hop MIS uniform protocols (Sect. V). Given a 2-hop coloring, we prove that the CONGEST model can be simulated with some multiplicative overhead (Sect. VI). Using this simulation and the algorithm proposed in [16] for the CONGEST model, we get an $O((a^2\Delta^2 + a^\mu\Delta^4) \cdot \log^2 n + a^3\Delta^3 \log n)$ time $O(a)$-coloring protocol in the beeping model, for any given positive constant $\mu < 1$. To the best of our

knowledge, this is the first coloring protocol using less than $\Delta + 1$ colors in the beeping model.

II. Ruling Set Protocol and Competition Graphs

Ruling sets serve as building blocks to construct complex protocols. They have been used to compute MIS [13] and colorings [17], [18]. In these papers, the ruling sets are used to decompose the network, and nodes in the ruling set (the "local leaders") take care of solving the problem for the nodes within a certain distance. In the beeping model, doing so is more difficult. We show in the next section, how ruling sets can still be used to design an efficient coloring protocol.

In this section, we introduce a competition protocol ($RulingSet$ - Protocol 1 in Sect. II-B) computing a $(2, 2\log N)$-ruling set. This protocol can be considered as a variant of the ruling set algorithm from [13]. That algorithm is heavily recursive, requiring concurrent communications, which are incompatible with the beeping model. Therefore, we adapt it and provide a non-recursive competition protocol with a similar behavior. To prove correctness (Sect. II-C), we use competition graphs, which are directed graphs that serve to model the behavior of competition protocols and help analyzing them. They were first used in [15], but in association with a non-deterministic elimination process. As we are interested in deterministic protocols, we use the nodes' identifiers to label the edges of a competition graph with α-encodings (Sect. II-A), and these values determine a deterministic elimination process. The resulting labeled competition graphs allow to compute the surviving nodes in a convenient way.

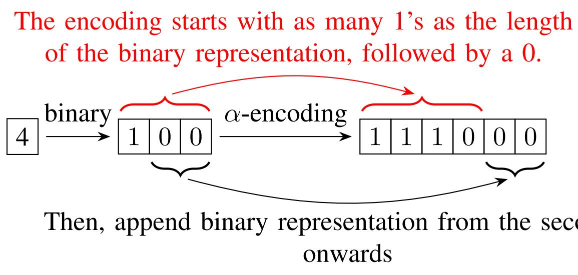

The encoding starts with as many 1's as the length of the binary representation, followed by a 0.
4
binary
1 0 0
α-encoding
1 1 1 0 0 0
Then, append binary representation from the second bit onwards

Fig. 3: Description of α-encoding

A. α-encoding

We α-encode integers to design *uniform* competition protocols, as in Casteigts *et al.* [19]. This encoding allows to compare integers (identifiers) bit by bit in a uniform manner. Indeed, when using α-encodings (of integers from $[N]$), such protocols do not need to know the binary integers' lengths (depending on $\log N$) to compare them bit by bit. The α-encoding of integer i is made up of two parts, as explained in Figure 3. Comparing two α-encodings $\alpha(i)$ and $\alpha(j)$ consists of first comparing the minimum number of bits necessary to encode the integers, and if it is the same, comparing the binary representations of i and j (the first bit is unnecessary, because it is 1 and the length is already known).

Definition 1. Let i be a positive integer, bin its binary representation and $l(i)$ the length of bin. Let bin_{-1} be the binary representation without the leading 1 and $1^{l(i)}$ a binary string of $l(i)$ 1's. The α-encoding of i, denoted by $\alpha(i)$, is $1^{l(i)} \parallel 0 \parallel bin_{-1}$ (where $\parallel$ denotes string concatenation).

α-encoding preserves the order between two integers.

Lemma 1. *For any $i, j \in \mathbb{N}^{>0}$: $i < j \Leftrightarrow \alpha(i) \prec \alpha(j)$, where $\prec$ is the lexicographical order on α-encodings.*

B. Uniform Competition Protocol for Computing a Ruling Set

Nodes use their unique identifiers for comparison and survivors of the elimination process constitute the output set. Each node v has a unique identifier $id(v)$. The identifiers are encoded on at most l_{max} bits, but l_{max} is unknown to the nodes and thus the binary representations of the identifiers do not necessarily have the same length. Every node v computes the α-encoding $\alpha(id(v))$ (or $\alpha(v)$ for short, by notation abuse) and outputs a boolean value $survived_v$. We prove that the output is a $(2, 2\log N)$-ruling set (Theorem 2).

Protocol 1 *RulingSet*

```
1:  IN: id: Integer   OUT: survived: Boolean value
2:  $survived := true$, $\alpha := \alpha(id)$                        ▷ Get $\alpha$-encoding
3:  for round $r := 1$ ; $r \leq l(\alpha)$ ; $r$++ do      ▷ $r$ is incremented after each iteration
4:      if $\alpha_r = 1$ then BEEP                ▷ Consider the $r^{th}$ most significant bit
5:      else
6:          LISTEN
7:          if beep heard then                     ▷ If a neighbor has a higher identifier
8:              $survived := false$
9:              EndProtocol
10: EndProtocol                                                    ▷ No beep heard
```

The following lemma is straightforward.

Lemma 2. *RulingSet has a round complexity of $max_{v \in V} l(\alpha(v)) = O(\log N)$.*

C. Correctness Analysis of Protocol 1

The output set of $RulingSet$ is analyzed through a game, which we refer to as the "elimination game". This game is enacted on an *edge-labeled directed acyclic graph* G_{dag}, the labeled competition graph, constructed from the original communication graph G and the nodes' unique identifiers. This construction process is adapted here to the $RulingSet$ protocol, but it applies to any competition protocol. $G_{dag} = (V, E_{dag}, label)$, where E_{dag} is the set of directed edges and $label$ an edge labeling function. G_{dag} is constructed from the α-encodings of the identifiers, encoded on a maximum of $2l_{max}$ bits.

- Let (u, v) be an edge of G with $\alpha(u) \succ \alpha(v)$. Then, (u, v) is a directed edge in G_{dag}, directed from u to v.
- Let (u, v) be an edge of G_{dag}. For the smallest index $i \in [2l_{max}]$ such that $\alpha_i(u) = 1$ and $\alpha_i(v) = 0$, set $label(u, v)$ to i: the edge (u, v) is labeled (with) i.

For any edge $e = (u, v)$ of G_{dag}, u is called the *origin* and v the *extremity* of e. The following lemma is straightforward.

The elimination game is played by the nodes of G_{dag}, round by round, in the following way: on round r, all surviving nodes

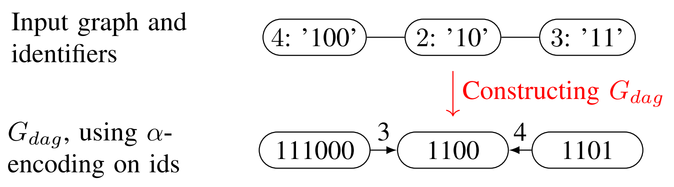

Input graph and identifiers
4: '100'
2: '10'
3: '11'
Constructing G_{dag}
G_{dag}, using α-encoding on ids
111000
1100
1101
3
4

Fig. 4: Example of G_{dag} construction

with an outwards edge e labeled $label(e) = r$ eliminate the extremities of these edges. The game finishes when no more node can be eliminated (thus after at most $2l_{max}$ rounds). A node's survival is stored as a boolean in the *survived* variable.

Definition 2. Let v be a vertex in G_{dag}. Let e be an incoming edge. We say that e is *acting* if the origin of e is not eliminated before round $label(e)$, and *non acting* otherwise. If $e = (u, v)$ is an acting incoming edge, then u eliminates v at round $label(e)$ if and only if v has not already been eliminated. We define the same notions for outgoing edges.

Definition 3. Let $\Pi = (v_1, \ldots, v_l)$ be a directed path in G_{dag}. There is a unique label sequence $S_{lab}(\Pi) = (s_1, \ldots, s_{l-1})$ s.t. $\forall r \in [l-1]$, $e_r = (v_r, v_{r+1})$ and $s_r = label(e_r)$.

Results similar to the following lemma and theorem are proven in [20] for a more limited case. Lemmas 3 and 4 are straightforward.

Lemma 3. *Let $\Pi = (v_1, \ldots, v_l)$ be a directed path in G_{dag}. $S_{lab}(\Pi)$ has no consecutive equal labels: $\forall r \in [l-1]$ $s_r \neq s_{r+1}$.*

Theorem 1. *Let v be any node from G_{dag} not surviving the elimination game. There exists a surviving node u such that $dist(u, v) \leq 2l_{max}$, where $l_{max} = O(\log N)$.*

Proof. First, a path Π from a surviving node u to node v is constructed, then we prove that Π's length is at most $2l_{max}$. Π is constructed by induction. Node v did not survive, so there exists an acting incoming edge. The acting incoming edge (w, v) with the smallest label is added to Π. If w does not survive the elimination game, the previous actions are repeated and an acting incoming edge is added to Π. This is done until a surviving node is reached. Since at least one node survives the elimination game, Π's construction is well-defined and $\Pi = (e_l, \ldots, e_1)$.
Now, let us prove by contradiction that $l \leq 2l_{max}$. Suppose $l > 2l_{max}$ and focus on $S_{lab}(\Pi)$. Because the edge-labels are integers from $[2l_{max}]$ and consecutive labels are non equal by Lemma 3, there exists an extremum s_r indexed by $r \in \{2, \ldots, 2l_{max}\}$. Thus there exists $i \in \{r-1, r\}$ such that $s_i > s_{i+1}$. However, both e_i and e_{i+1} are acting incoming edges, by construction. Thus, the origin of e_i is eliminated in round s_{i+1}, which contradicts the fact that e_i is acting. Hence, we have a contradiction. $\square$

Lemma 4. *Let $I = \{v \in V$ s.t. $survived_v = true\}$ at the termination of RulingSet. Let S be the set of survivor nodes of an elimination game played on G_{dag}. We have $I = S$.*

Theorem 2. *The output set $I = \{v \in V$ s.t. $survived_v = true\}$ of RulingSet is a $(2, 2\log N)$-ruling set.*

III. MIS and Vertex Coloring Protocols

Let us now present MIS and $(\Delta+1)$-coloring protocols with $O(\Delta^2 \log N + \Delta^3)$ round complexity, where Δ is the maximum degree of the communication graph G. When $\Delta = O(1)$, we obtain an asymptotically optimal $O(\log n)$ round complexity [10] . For polylogarithmic Δ, the protocol is still very efficient. Nodes know the maximum degree Δ at first, but this assumption is dropped later on. Nodes know no polynomial upper bound N on their total number.

The protocols presented here are based on computing and refining defective colorings. Defective colorings were first used to solve the distributed coloring problem in [21] and [22]. Here, we refine the defective coloring differently from the previous works. The exact method is explained below.

A. Non-uniform Protocols for MIS and $(\Delta+1)$-coloring

The $(\Delta+1)$-coloring protocol $DegreeColoring$ (Protocol 2) refines the initial Δ-defective coloring until the coloring is proper. $DegreeColoring$ can be seen as an $\mathcal{L}_4$-phase. It has Δ $\mathcal{L}_3$-phases: in each of these phases, the defect is reduced by at least 1. Each $\mathcal{L}_3$-phase is made of $\Delta+1$ *coloring* $\mathcal{L}_2$-phases, followed by an additional *color reduction* $\mathcal{L}_2$-phase.
In each coloring $\mathcal{L}_2$-phase, nodes with a specific color compute a $(2, 2\log N)$-ruling forest on the subgraph induced by themselves, using $RulingSet$ (Protocol 1) and Breadth First Searches (BFS) - $ColorByBFS$ function (see below). During BFS, they recolor themselves with an even or odd available color depending on the parity of their depth in the ruling forest. Finally, all nodes communicate the changes in color and update their set of unavailable colors. The following color reduction $\mathcal{L}_2$-phase is made up of $\Delta+1$ $\mathcal{L}_1$-phases. In each such $\mathcal{L}_1$-phase, the range of colors used by all nodes is reduced by 1 (if the range is greater than $\Delta+1$). This is important because the color range affects the round complexity, and the color range can increase exponentially if it is not reduced in each $\mathcal{L}_3$-phase.

Let us now present the functions used in a coloring $\mathcal{L}_2$-phase. There are two functions, $ColorByBFS$ and $BroadcastColors$. $ColorByBFS$ recolors each participating node. The resulting coloring can be defective. The input parameters are a boolean ($inSet$) indicating whether or not the node is part of the ruling set, i.e., serving as BFS roots, and a set of unavailable colors (U). The roots initiate parallel BFS. Other nodes compute their distance to the nearest root, which is given by the BFS, and recolor themselves with an available (not in U) $newColor$, according to the parity of this distance. The $newColor$ values returned by $ColorByBFS$ are in $[2\Delta+2]$. This is because the set of unavailable colors contains at most Δ colors, and possibly all of the same parity. Therefore, $ColorByBFS$ chooses the smallest available odd (resp. even) value amidst the first $\Delta+1$ odd (resp. even) values. $BroadcastColors$ communicates the colors chosen by the neighboring nodes. The function has four input parameters: the

function ColorByBFS($inSet$, U): $newColor$

```
1: $beeping := false$                                  ▷ $|U| \leq \Delta$, $U \subset [2\Delta + 2]$.
2: if $inSet$ then $beeping := true$                   ▷ Roots initiate BFS
3: for round $r := 1$ ; $r$++ do
4:     if $beeping$ then
5:         BEEP              ▷ All nodes beep once. The root is $r - 1$ hops away.
6:         $newColor := min\{k \in (\mathbb{N}^{>0} \setminus U) \mid k \equiv r \mod 2\}$
7:         Return $newColor$                           ▷ $newColor \in [2\Delta + 2]$
8:     else
9:         LISTEN
10:        if beep heard then $beeping := true$
```

node's color ($newColor$), a boolean indicating whether or not it should participate in the current invocation ($changingColor$), a set of unavailable colors (U) and the maximum degree Δ. The color is transmitted through the round number.

function BroadcastColors($newColor$, $changingColor$, U, Δ): U

```
1: for round $r := 1$ ; $r \leq 2\Delta + 2$ ; $r$++ do
2:     if $newColor = r$ and $changingColor$ then BEEP
3:     else
4:         LISTEN
5:         if beep heard then $U := U \cup \{r\}$      ▷ More unavailable colors
6: Return $U$
```

Now, let us present the $ColorReduction$ function invoked in the color reduction $\mathcal{L}_2$-phase. Its input parameters are an integer value given by a d-defective c-coloring ($color$), the maximum degree Δ and the maximum color c (in the coloring). Nodes broadcast colors from $[\Delta + 1]$ in $\Delta + 1$ rounds, after which, nodes with $color = c$ change their color to the smallest available color in $[\Delta + 1]$. The output is the node's new color ($color$), given by a d-defective c'-coloring, with $c' = min(c - 1, \Delta + 1)$.

function ColorReduction($color$, Δ, c): $color$

```
1: $U := \emptyset$                                    ▷ $U$ stores unavailable colors
2: for round $r := 1$ ; $r \leq \Delta + 1$ ; $r$++ do
3:     if $color = r$ then BEEP
4:     else
5:         LISTEN
6:         if beep heard then $U := U \cup \{r\}$
7: if $color = c$ then $color := min([\Delta + 1] \setminus U)$
8: Return $color$
```

In $DegreeColoring$, we only require $\mathcal{L}_1$-synchronization points (for $RulingSet$ and $ColorByBFS$), introduced in Sect. I-C. Both functions are uniform in N, and thus are not explicitly terminating (if executed alone). However, they are locally terminating. Therefore we can use BET to perform neighboring termination detection and make nodes start the next step of the protocol synchronously. On the other hand, as the time lengths of all $\mathcal{L}_2$, $\mathcal{L}_3$ and $\mathcal{L}_4$-phases (and $ColorReduction$ calls) are upper bounded by $\Delta + 1$, their termination is completely synchronized at all nodes and we do not need $\mathcal{L}_2$, $\mathcal{L}_3$ and $\mathcal{L}_4$-synchronization points.

Lemma 5. *At the start (and end) of an $\mathcal{L}_3$-phase, color $\in [\Delta + 1]$.*

Lemma 6. *The defect of color is reduced by one per $\mathcal{L}_3$-phase.*

Proof. Let $color$ be d-defective at the start of $\mathcal{L}_3$-phase $p3$. For any given node v, v has at most d defective edges. It is

Protocol 2 $DegreeColoring$

```
1: IN: $id$: Identifier, $\Delta$: Maximum degree   OUT: $color$: Integer value
2: $color := 1$
3: // Each node removes at least one defective edge per $\mathcal{L}_3$-phase
4: for $\mathcal{L}_3$-phase $p3 := 1$ ; $p3 \leq \Delta$ ; $p3$++ do
5:     $U := \emptyset$        ▷ Stores $newColor$ values already chosen during this phase
6:     $newColor := 0$                   ▷ $newColor \in [2\Delta + 2]$ during $\mathcal{L}_3$-phase
7:     // An $\mathcal{L}_3$-phase starts with $\Delta + 1$ coloring $\mathcal{L}_2$-phases
8:     for coloring $\mathcal{L}_2$-phase $p2 := 1$ ; $p2 \leq \Delta + 1$ ; $p2$++ do
9:         if $color = p2$ then $inSet := RulingSet(id)$
10:        $\lightning_1$                                ▷ $\mathcal{L}_1$-synchronization point
11:        if $color = p2$ then $newColor := ColorByBFS(inSet, U)$
12:        $\lightning_1$
13:        $U := BroadcastColors(newColor, (color = p2), U, \Delta)$
14:        $\lightning_2$      ▷ This synchronization point is not needed (strictly explanatory)
15:    $color := newColor$
16: // Followed by one $\mathcal{L}_2$-phase, which contains $\Delta + 1$ color reduction $\mathcal{L}_1$-phases
17: // Before the $\mathcal{L}_2$-phase, $color \in [2\Delta + 2]$
18: for color reduction $\mathcal{L}_1$-phase $p1 := 1$ ; $p1 \leq \Delta + 1$ ; $p1$++ do
19:     $color := ColorReduction(color, \Delta, 2\Delta + 3 - p1)$
20:     $\lightning_1$
21: // After all color reduction $\mathcal{L}_1$-phases, $color \in [\Delta + 1]$
22: $\lightning_3$                                       ▷ Not needed, for clarity only
23: EndProtocol
```

easy to see that non-defective edges remain non-defective. In a non-defective edge (v, u), let v be the node with the higher color w.l.o.g. During $\mathcal{L}_3$-phase $p3$, v stores a set of unavailable $newColor$ values, including $newColor_u$. As such, when v executes $ColorByBFS$, $newColor_v \neq newColor_u$.
All endpoints of the defective edges of v, and v itself, execute $RulingSet$ and $ColorByBFS$ in the same $\mathcal{L}_2$-phase. If $DistC(v)$ denotes v's distance to the nearest BFS tree root (RulingSet survivor), there is at least one endpoint u with $|DistC(u) - DistC(v)| = 1$. Because of the difference in the parity of these distances, u and v choose different values in $[2\Delta + 2]$, and at least one edge becomes non-defective. ☐

Theorem 3. *$DegreeColoring$ solves $(\Delta + 1)$-coloring in $O(\Delta^2 \log n + \Delta^3)$ rounds.*

Given a $(\Delta + 1)$-coloring, it is simple to compute an MIS in $\Delta + 1$ rounds. Nodes with the same color form an independent set. Adding iteratively (at each round) nodes from each such set to a common independent set results in an MIS. Thus, MIS can also be solved in $O(\Delta^2 \log n + \Delta^3)$ rounds.

B. Uniform $(\Delta + 1)$-coloring

Now, we wish to transform $DegreeColoring$ into $UnifDegreeColoring$, which is uniform in both Δ and n. The first step is to replace the functions used in $DegreeColoring$ by uniform functions, and to synchronize them using synchronization points. Then, every non-uniform stopping condition of a loop appearing in $DegreeColoring$ should be eliminated and replaced by a so called *local termination component*. This *component* is an $\mathcal{L}_{i-2}$-phase executed at the end of each iteration ($\mathcal{L}_{i-1}$-phase) of the loop ($\mathcal{L}_i$-phase). It serves to detect if the executing node has finished the ongoing loop. More formally, this component serves to detect whether the executing node has reached a terminal $\mathcal{L}_i$-phase state, and makes the $\mathcal{L}_i$-phase locally-terminating.

First, let us present $UnifBroadcastColors$, a uniform version of $BroadcastColors$ (since $BroadcastColors$ requires Δ). $UnifBroadcastColors$ is an $\mathcal{L}_2$-phase, made of consecutive $\mathcal{L}_1$-phases, each composed of 2 rounds. In the first round, the executing node v beeps if it has not yet communicated $newColor_v$. Otherwise, it listens so it can detect if all of its neighbors have already communicated their $newColor$ value, and if so, v terminates. In the second round, we have the round behavior of $BroadcastColors$. In such a way, we obtain a uniform function having the same behavior as $BroadcastColors$. Moreover, in this particular case, since all $\mathcal{L}_1$-phases contain exactly 2 rounds, it is also locally synchronized, even without using EBET, and therefore there is no need to indicate synchronization points explicitly.

```
function UnifBroadcastColors($newColor$, $changingColor$, $U$): $U$
for $\mathcal{L}_1$-phase $p1 := 1$ ; $p1$++ do          ▷ $\mathcal{L}_1$-phase consists of two rounds
    // First round
    if $newColor >= p1$ and $changingColor$ then BEEP ▷ Not finished yet
    else
        LISTEN
        if no beep heard then Return $U$       ▷ If all neighbors beeped their colors
    // Second round
    if $newColor = p1$ and $changingColor$ then
        BEEP                                   ▷ Communicate your color
    else
        LISTEN
        if beep heard then $U := U \cup \{p1\}$ ▷ Keep neighbors' $newColor$ values
```

Next, we design a uniform version of $ColorReduction$. It is used in $ReduceColors$, a uniform version of the color reduction $\mathcal{L}_2$-phase from $DegreeColoring$.
$UnifColorReduction$ has two input parameters: the node's color (*color*), given by a d-defective c-coloring, and a set of unavailable colors (U). It also has two output parameters: the node's new color *color*, given by a d-defective c'-coloring (with $c' = min(c-1, \Delta+1)$), and a boolean $sameColor$ indicating whether *color* changed. Every node v transmits its *color* value to its neighbors by beeping in the first round of the $\mathcal{L}_1$-phase indexed by *color*. Nodes with the highest color in their neighborhood choose the smallest available color (colors previously transmitted by neighbors are forbidden). If that color is the node's current color, then $sameColor$ is assigned to true. Other nodes do not change their color (and end with $sameColor$ equal to false). Here again, there is no need to indicate synchronization points explicitly, since all $\mathcal{L}_1$-phases contain exactly 2 rounds.
$ReduceColors$ is an $\mathcal{L}_4$-phase. It has two input parameters: the node's color (*color*), given by a d-defective c-coloring, and a set of unavailable colors (U). It has a single output: the node's new color (*color*), given by a d-defective $(\Delta+1)$-coloring. The main idea is to have the nodes with the highest color in their neighborhood change their color to the smallest available color (in $[\Delta+1]$). At some point, they can no longer improve their color ($finished$ is true). These nodes terminate, allowing the other nodes in their neighborhood to change their *color* value. Here, it is crucial to put $\mathcal{L}_2$-synchronization points after the $UnifColorReduction$ and $UnifBroadcastColors$ calls, because these functions are uniform. Thus, different nodes can finish executing these functions at different times, i.e., not synchronously. As these functions are locally terminating, EBET can be used to ensure the synchronization points.

```
function UnifColorReduction($color$, $U$): $color$, $sameColor$
$sameColor := false$
for $\mathcal{L}_1$-phase $p1 := 1$ ; $p1$++ do          ▷ $\mathcal{L}_1$-phase consists of two rounds
    // First round
    if $color = p1$ then BEEP
    else
        LISTEN
        if beep heard then $U := U \cup \{p1\}$
    // Second round
    if $color > p1$ then BEEP
    else
        // Only a node with the highest color in its neighborhood hears no beep
        LISTEN
        if beep heard then Return ($color$, $sameColor$)
        else
            $color := min([p1] \setminus U)$
            if $color$ = $p1$ then $sameColor := true$
            Return ($color$, $sameColor$)
```

```
function ReduceColors($color$, $U$): $color$
$finished := false$
while not $finished$ do                            ▷ At most $c$ $\mathcal{L}_3$-phases
    ($color$, $finished$) $:= UnifColorReduction(color, U)$
    $↯_2$
    $U := UnifBroadcastColors(color, finished, U)$
    $↯_2$                         ▷ Actually, also an $\mathcal{L}_3$-synchronization point
Return $color$
```

Following this, let us describe the functions used for $UnifDegreeColoring$'s local termination component. These functions are used to detect when the executing node's color is proper, i.e., no neighbor has the same color. Then, the executing node can exit the outermost loop and thus locally terminate the protocol (see lines 24 to 29).
$ColorCollision$ uses $UniformCollisonBeep$ to detect whether there are same color neighbors amongst executing nodes. The function has two input parameters: an identifier (*id*) and the node's color (*color*). The output parameter is a boolean indicating whether the node detected a collision with a same color node (*collision*). In each $\mathcal{L}_2$-phase $p2$, nodes with color $p2$ check for a collision by using $UnifCollisionBeep$. If no neighboring node with the same color $p2$ exists, then no collision is detected by the executing nodes.
$UnifCollisionBeep$ detects whether there are any neighbors amongst the currently executing nodes (a *collision*). The input parameter is an identifier (*id*) and the output parameter is a boolean indicating whether the node detected a collision (*collision*). In each $\mathcal{L}_1$-phase, a node beeps in the first or the second round, depending on whether the $p1^{th}$ most significant bit of $\alpha(id)$ is 0 or 1. If a beep is heard, then there is a collision. Two executing neighboring nodes always detect a collision because they have different identifiers. A node terminates if the phase index $p1$ is greater than the length of the α-encoding of its id.

Finally we describe $UnifDegreeColoring$. The main idea is the same as in $DegreeColoring$: we refine the initial Δ-defective coloring until the coloring is proper. The main differences are the local termination components. The $\mathcal{L}_4$-phase's ($\mathcal{L}_3$ loop) local termination component is similar to the local termination component in $UnifBroadcastColors$. A node has finished an $\mathcal{L}_4$-phase if all of its neighbors have

```
**function** UnifCollisionBeep($id$): $collision$
$collision := false$
**for** $\mathcal{L}_1$-phase $p1 := 1$ ; $p1$++ **do**        ▷ $\mathcal{L}_1$-phase consists of two rounds
    **if** $p1 > l(\alpha(id))$ **then** Return $collision$
    **if** $\alpha_{p1}(id) = 0$ **then**
        BEEP ; LISTEN
        **if** beep heard in the second round **then** $collision := true$
    **else**
        LISTEN ; BEEP
        **if** beep heard in the first round **then** $collision := true$
```

```
**function** ColorCollision($id$, $color$): $collision$
**for** $\mathcal{L}_2$-phase $p2 := 1$ ; $p2$++ **do**        ▷ At most $\Delta + 1$ $\mathcal{L}_2$-phases
    **if** $color = p2$ **then**
        $collision := UnifCollisionBeep(id)$
        Return $collision$
    $\lightning_2$
```

chosen a new color. The protocol's local termination component is described previously. The additional U_t variable is used to store unavailable colors that have already been chosen by neighboring nodes which have terminated the protocol.

```
**Protocol 3** $UnifDegreeColoring$
**IN:** $id$: Identifier   **OUT:** $color$: Integer value
$U_t := \emptyset$        ▷ Stores $color$ values chosen for output by terminated neighbors
$color := 1$
// Each node removes at least one defective edge per $\mathcal{L}_5$-phase
**for** $\mathcal{L}_5$-phase $p5 := 1$ ; $p5$++ **do**
    $U := \emptyset$        ▷ Stores $newColor$ values already chosen during this phase
    $newColor := 0$        ▷ $newColor \in [2\Delta + 2]$ during the $\mathcal{L}_5$-phase
    // This $\mathcal{L}_3$ loop is an $\mathcal{L}_4$-phase
    **for** coloring $\mathcal{L}_3$-phase $p3 := 1$ ; $p3$++ **do**
        **if** $color = p3$ **then** $inSet := RulingSet(id)$
        $\lightning_1$
        **if** $color = p3$ **then** $newColor := ColorByBFS(inSet, U \cup U_t)$
        $\lightning_1$
        $U := UnifBroadcastColors(newColor, color = p3, U)$
        $\lightning_2$        ▷ $\mathcal{L}_2$-synchronization point here, thus we have coloring $\mathcal{L}_3$-phases
        // From here on, local termination component for $\mathcal{L}_4$-phase
        **if** $color > p3$ **then** BEEP        ▷ Beep if new color still not chosen
        **else**
            LISTEN        ▷ Check if any neighbor is still choosing a new color
            **if** no beep heard **then** Exit $\mathcal{L}_3$ loop
    $\lightning_4$        ▷ Crucial because some nodes end the $\mathcal{L}_4$-phase earlier than others
    $color := ReduceColors(newColor, U_t)$        ▷ After, $color \in [\Delta + 1]$
    $\lightning_4$
    // From here on, local termination component for $\mathcal{L}_5$ loop
    $collision := ColorCollision(color)$
    $\lightning_3$        ▷ Because $ColorCollision$ is an $\mathcal{L}_3$-phase
    $U_t := UnifBroadcastColors(color, collision = false, U_t)$
    $\lightning_2$        ▷ Because $UnifBroadcastColors$ is an $\mathcal{L}_2$-phase
    **if** not $collision$ **then** EndProtocol        ▷ Exit $\mathcal{L}_5$ loop
```

Theorem 4. *MIS and $(\Delta + 1)$-coloring can be solved in $O(\Delta^2 \log n + \Delta^3)$ rounds with a protocol uniform in both Δ and N.*

IV. Improvements for graphs with small arboricity

$DegreeColoring$ is efficient for graphs with polylogarithmic maximum degree Δ. However, not all graphs have a low maximum degree, and in these graphs, Protocol 3 is less efficient. Using ideas from [23] and [24], it is possible to design a $(\Delta+1)$-coloring protocol which is efficient on graphs with low arboricity a (more specifically, with polylogarithmic a). Notice that some important topologies like trees and planar graphs have an arboricity of 1 and 3 respectively, while their maximum degree can be arbitrarily large.

Theorem 5. *MIS and $(\Delta + 1)$-coloring can be solved with $O(a^2 \log^2 n + a^3 \log n)$ round complexity in the beeping model, where a is the arboricity of the communication graph.*

To support this theorem, we design two coloring protocols with the above round complexity: one is uniform in N but not in a, and the other is uniform in a but not in N. It is important to have a protocol uniform in a, since a may be harder to obtain than an upper bound on N. The following results from [24] are used to obtain these protocols.

Lemma 7. *[24] If G is of arboricity a, at least $\frac{\epsilon}{2+\epsilon}|V|$ nodes have a degree less than $(2 + \epsilon)a$.*

Theorem 6. *[24] If G is of arboricity a, it can be decomposed into $l = O(\log n)$ sets of nodes $H_1, \ldots, H_l$ such that each set H_i has maximum degree $O(a)$ in the induced subgraph $G[\cup_{k=i}^{l} H_k]$.*

The $LimitedDegreeColoring$ function is the main component of both protocols. It colors all participating low-degree nodes, if it is given an upper bound on the arboricity a. A node v is considered to be a low-degree node if it has $deg(v) \leq \Delta_a$, where $\Delta_a = (2 + \epsilon) \cdot a$ for a parameter $\epsilon > 0$. Contrarily to $DegreeColoring$, it may happen that some nodes have no available colors in $[\Delta_a + 1]$, due to their high degree, and end the function uncolored, represented by the color 0. We use $LimitedColorReduction$, a slightly modified version of $ColorReduction$. It is not presented here, but the only change is that $color$ is set to 0 if $[\Delta + 1] \setminus U$ is an empty set.

```
**function** LimitedDegreeColoring($id$, $c$): $color$
$color := 1$
**for** $\mathcal{L}_3$-phase $p3 := 1$ ; $p3 \leq c$ ; $p3$++ **do**
    $U := \emptyset$        ▷ Stores $newColor$ values already chosen during this phase
    $newColor := 0$
    **for** $\mathcal{L}_2$-phase $p2 := 1$ ; $p2 \leq c + 1$ ; $p2$++ **do**
        **if** $color = p2$ **then** $inSet := RulingSet(id)$
        $\lightning_1$        ▷ $\mathcal{L}_1$-synchronization point
        **if** $color = p2$ **then**
            $newColor := ColorByBFS(inSet, U)$
            **if** $color \notin [2c + 2]$ **then** Return 0        ▷ Not a good color
        $\lightning_1$
        $U := BroadcastColors(newColor, color = p2, U, c)$
    $color := newColor$
    **for** $\mathcal{L}_1$-phase $p1 := 1$ ; $p1 \leq c + 1$ ; $p1$++ **do**
        $color := LimitedColorReduction(color, c, 2c + 3 - p1)$
        **if** $color = 0$ **then** Return 0        ▷ No color in $[c + 1]$ could be chosen
$collision := ColorCollision(color)$
**if** $collision$ **then** Return 0        ▷ If not properly colored, no color is chosen
Return $color$
```

Lemma 8. *Let $\Delta_a = (2 + \epsilon) \cdot a$, with $\epsilon > 0$. Given the input $c = \Delta_a$, LimitedDegreeColoring outputs a $(\Delta_a + 1)$-coloring on a subgraph of nodes, which includes all nodes with degree less than or equal to Δ_a. All other nodes have output 0. The round complexity is $O(a^2 \cdot \log n + a^3)$.*

Proof. The round complexity is straightforward. $LimitedDegreeColoring$ outputs a $(\Delta_a + 1)$-coloring on the subgraph of nodes with non-zero colors because all colors are chosen from $[2\Delta_a + 2]$, if available. and are then reduced to $[\Delta_a + 1]$. $ColorCollision$ ensures that the coloring is valid.

Now, let us prove by contradiction that for any given node u with $deg(u) \leq \Delta_a$, the output is a non-zero color. u outputs 0 due to $LimitedColorReduction$, $ColorByBFS$ or $ColorCollision$. The first two cases are impossible because $|U(u)| \leq \Delta_a$. In the last case, $ColorCollision$ is executed after Δ_a $\mathcal{L}_3$-phases. In each $\mathcal{L}_3$-phase, incident non-defective edges remain non-defective, and at least one incident defective edge becomes non-defective. Since after Δ_a $\mathcal{L}_3$-phases u has no defective edges, u has no neighbor v with $color_u = color_v$. □

A. $(\Delta + 1)$-coloring Uniform in N

First, let us focus on the *first protocol*, uniform in N. $LimitedDegreeColoring$ is executed iteratively by uncolored nodes until all nodes are properly colored. Since a is known, and by Lemma 7, each invocation of the function colors a constant fraction of the nodes of the communication graph. Colored nodes no longer participate in subsequent $LimitedDegreeColoring$ calls. By Theorem 6, executing $LimitedDegreeColoring$ $l = O(\log N)$ times (or more) colors all nodes with $O(a \cdot \log N)$ colors. As N is unknown, invocations of $LimitedDegreeColoring$ continue until the executing node is colored properly (local termination component). When this happens for all nodes, the $O(a \cdot \log N)$-coloring is transformed into a $(\Delta + 1)$-coloring by $ReduceColors$, as in Protocol 3. This takes an additional $O(a^2 \cdot \log^2 n)$ rounds.

Protocol 4 $UnifNArbColoring$

```
IN: id: Identifier, a: Arboricity of G, ε: Parameter
OUT: color: Integer value
$\Delta_a := (2+\epsilon) \cdot a$
for $\mathcal{L}_4$-phase $p4 := 1$ ; $p4$++ do                ▷ At most $l = \frac{2}{\epsilon} \cdot \log n$ $\mathcal{L}_4$-phases
    $color := LimitedDegreeColoring(id, \Delta_a)$
    ↯$_4$
    if $color \neq 0$ then
        $color := color + (p4 - 1) \cdot (\Delta_a + 1)$
        Exit $\mathcal{L}_4$ loop
↯$_5$                                                  ▷ $color$ is an $O(a \cdot \log n)$-coloring
$color := ReduceColors(color, \emptyset)$                 ▷ At most $O(a^2 \cdot \log^2 n)$ rounds
EndProtocol                                            ▷ $color \in [\Delta + 1]$
```

Theorem 7. *Protocol 4 solves MIS and $(\Delta + 1)$-coloring with $O(a^2 \log^2 n + a^3 \log n)$ round complexity. This protocol is uniform in N but non-uniform in a.*

Proof. Let us prove that after all $\mathcal{L}_4$-phases, $arbColor$ is an $O(a \cdot \log n)$-coloring. In each $\mathcal{L}_4$-phase of Protocol 4, only uncolored nodes (V_{rem}) participate in $LimitedDegreeColoring$. Since the subgraph induced by V_{rem} also has arboricity at most a, by Lemmas 7 and 8, $\frac{\epsilon}{2+\epsilon}|V_{rem}|$ nodes have a degree less than Δ_a and thus are part of the subgraph with a $(\Delta_a + 1)$-coloring. They exit the $\mathcal{L}_4$ loop, thus by Theorem 6, there are at most $\frac{2}{\epsilon} \cdot \log n = O(\log n)$ $\mathcal{L}_4$-phases. Since we use non-overlapping ranges of $\Delta_a + 1$ colors for each $\mathcal{L}_4$-phase, $arbColor$ is an $O(a \cdot \log n)$-coloring. The round complexity follows from the number of $\mathcal{L}_4$-phases and Lemma 8. □

B. $(\Delta + 1)$-coloring Uniform in a

In the *second protocol* (uniform in a), we compute an upper bound on a. This is done by estimating a iteratively. At each iteration ($\mathcal{L}_5$-phase) i, a is estimated to be 2^i and $LimitedDegreeColoring$ is executed $l = O(\log N)$ times, given this estimation. After $O(\log a)$ iterations, the estimation is at least as large as the actual arboricity. When this happens, $LimitedDegreeColoring$ executed $O(\log N)$ times provides a proper coloring (followed by the color range reduction) as in the first protocol.

Protocol 5 $UnifAArbColoring$

```
IN: id: Identifier, N: Polynomial upper bound on n, ε: Parameter
OUT: color: Integer value
for $\mathcal{L}_5$-phase $p5 := 1$ ; $p5$++ do              ▷ At most $1 + \lfloor \log a \rfloor$ $\mathcal{L}_5$-phases.
    $\Delta_{p5} := (2+\epsilon) \cdot 2^{p5}$
    $len := \frac{2}{\epsilon} \cdot \log N$
    for $\mathcal{L}_4$-phase $p4 := 1$ ; $p4 \leq len$ ; $p4$++ do
        $color := LimitedDegreeColoring(id, \Delta_{p5})$
        ↯$_4$
        if $color \neq 0$ then
            $color$ +=$(\Delta_{p5} - 2 - \epsilon + p5 - 1) \cdot len + (p4 - 1) \cdot (\Delta_{p5} + 1)$
            Exit $\mathcal{L}_5$ loop
    ↯$_5$                                              ▷ Not needed, for clarity only
↯$_6$                                                  ▷ color is an $O(a \cdot \log n)$-coloring
$color := ReduceColors(color)$                          ▷ At most $O(a^2 \cdot \log^2 N)$ rounds
EndProtocol                                            ▷ $color \in [\Delta + 1]$
```

Theorem 8. *Protocol 5 solves MIS and $(\Delta + 1)$-coloring with $O(a^2 \log^2 n + a^3 \log n)$ round complexity. This protocol is uniform in arboricity a but non-uniform in N.*

Proof. Let us first prove that Protocol 5 solves $(\Delta+1)$-coloring. At the end of $\mathcal{L}_5$-phase $p5$, by Lemma 8, all nodes with degree less than $\Delta_{p5} = (2+\epsilon) \cdot 2^{p5}$ are colored. By Lemmas 7 and 8, and Theorem 6, at $\mathcal{L}_5$-phase $p5 = 1 + \lfloor \log a \rfloor$, $\Delta_{p5} \geq (2+\epsilon) \cdot a$ and all nodes are colored after $\frac{2}{\epsilon} \cdot \log N$ $\mathcal{L}_4$-phase. Since the color ranges from different $\mathcal{L}_4$-phases or different $\mathcal{L}_5$-phases do not overlap, and the non-zero colors returned by $LimitedDegreeColoring$ form a coloring, $arbColor$ is an $O(a \cdot \log N)$-coloring. And after the $ReduceColors$ call, the coloring is reduced to a $(\Delta + 1)$-coloring.
It is straightforward to prove the round complexity. Since $arbColor$ is an $O(a \cdot \log N)$-coloring, $ReduceColors$ takes at most $O(a^2 \log^2 N)$ rounds, while the $\mathcal{L}_5$ loop takes at most $O(a^2 \log^2 N + a^3 \log N)$ rounds. □

V. Uniform Protocols for 2-hop MIS and 2-hop $(\Delta^2 + 1)$-coloring

To obtain protocols for 2-hop MIS and 2-hop coloring, we provide and use a general transformer, the $SquareSim$ protocol (Protocol 6), allowing to "simulate G^2 over G". The idea is that nodes propagate beeps for an extra round (and therefore contact nodes at distance 2), so that they can simulate a protocol on the square of the communication graph, for a small time multiplicative overhead. $SquareSim$ provides two primitives $SquareSim(true)$ and $SquareSim(false)$ to simulate in G, the $BEEP$ and $LISTEN$ instructions invoked on graph G^2.

Lemma 9. *A protocol designed to be executed on G^2 can be simulated on G by replacing all $BEEP$ instructions by calls to $SquareSim(true)$ and $LISTEN$ instructions by calls to $SquareSim(false)$.*

The maximum degree of the square communication graph is Δ^2. By applying Lemma 9 to the previous protocols, we obtain

Protocol 6 Simulating the square communication graph: $SquareSim$

```
 1: IN: beep: Boolean value   OUT: detectedBeep: Boolean value
 2: $detectedBeep := false$
 3: if $beep$ then BEEP                 ▷ Transmit beep to neighbor nodes: First round
 4: else
 5:     LISTEN
 6:     if beep heard then $detectedBeep := true$
 7: if $detectedBeep$ then BEEP         ▷ Transmit to distance 2 nodes: Second round
 8: else
 9:     LISTEN
10:     if beep heard and not $beep$ then $detectedBeep := true$
11: EndProtocol
```

protocols for solving 2-hop coloring with (Δ^2+1) colors and 2-hop MIS. These protocols are very efficient on bounded degree graphs, and efficient for graphs with polylogarithmic Δ. 2-hop coloring is an important tool in the beeping model, used to break symmetry and to deal with the interferences. In the next section, we show how this can be used to simulate the stronger CONGEST communication model and obtain an $O(a)$-coloring.

Corollary 1. *2-hop MIS and 2-hop (Δ^2+1)-coloring can be solved in $O(\Delta^4 \log n + \Delta^6)$ rounds.*

Instead of the maximum degree of the square of the given graph, consider its arboricity. Using a result from [25], showing that $a(G^2) \leq 2^3 \cdot a \cdot \Delta$, we obtain Corollary 2, which provides a more efficient result for graphs with small arboricity.

Corollary 2. *2-hop MIS and 2-hop (Δ^2+1)-coloring are solved by the two protocols in Sect. IV with an $O(a^2\Delta^2\log^2 n + a^3\Delta^3\log n)$ round complexity. One of them is uniform in N but not in a, and the other is uniform in a but not in N.*

VI. CONGEST Model Simulation and $O(a)$-coloring

By using a 2-hop coloring, nodes can simulate the transmission of messages through the edges of the communication graph, like in the CONGEST model with edge bandwidth B (commonly $O(\log N)$). We want to make sure that for any given node v, a message can be sent or received along any edge without interference, and that the provenance and destination of the message can be deduced easily.

First, $InitCongest$ (Protocol 7) is used at the beginning of the simulation to obtain all possible message provenance and destinations for any given node v (simulated by the colors from the 2-hop coloring). After which, the transmission of messages is done through $SimCongest$.

Our simulation algorithm $SimCongest$ (Protocol 8) is made of two components. The *first component* is used to transmit a B bit message. If we have no interference, a node can transmit B bits during $2B$ rounds (in phases of two rounds, one round for transmitting bit 1 and another one for bit 0).
The *second component*, and the core part of the simulation, deals with the interferences inherent to the beeping model. Here, a 2-hop c-coloring (for some constant c) is required so that messages can be associated to a pair of colors $p = (colorProvenance, colorDestination)$, according to their provenance and destination (c^2 possibilities). The simulation is composed of phases, each of c^2 invocations of the first component. In this way, transmitted bits never collide. The B bit messages are part of the input parameters of $SimCongest$. They are given through a hash table ($mSend$), with the message destinations (colors) as keys and the messages as values. The messages received are stored in a similar structure ($mRec$), where the message provenances are the keys.

Protocol 7 $InitCongest$

```
1: IN: color: Integer value from a 2-hop $c$-coloring
2: OUT: Nb: Port numbers
3: $Nb := \emptyset$                          ▷ Stores neighbors' colors (used as ports)
4: for round $r := 1$ ; $r \leq c$ ; $r$++ do          ▷ Get neighbors' colors
5:     if $r = color$ then BEEP
6:     else
7:         LISTEN
8:         if beep heard then $Nb := Nb \cup \{r\}$
9: EndProtocol
```

Protocol 8 $SimCongest$

```
 1: IN: B: Edge bandwidth, color: Integer value from a 2-hop $c$-coloring, $c$: maximum
    color value, mSend: Hash table of messages to send
 2: OUT: mRec: Hash table of messages received
 3: for $\mathcal{L}_3$-phase $p3 := 1$ ; $p3 \leq c$ ; $p3$++ do
 4:     for $\mathcal{L}_2$-phase $p2 := 1$ ; $p2 \leq c$ ; $p2$++ do
 5:         for $\mathcal{L}_1$-phase $p1 := 1$ ; $p1 \leq B$ ; $p1$++ do
 6:             if $p3 = color$ then                ▷ $p3$ can send its $p1$th bit to $p2$
 7:                 if $p2 \in Nb$ and $mSend[p2]_{p1} = 0$ then    ▷ Send a 0 message
 8:                     BEEP ; LISTEN
 9:                 else if $p2 \in Nb$ and $mSend[p2]_{p1} = 1$ then ▷ Send a 1 message
10:                     LISTEN ; BEEP
11:                 else
12:                     LISTEN ; LISTEN                          ▷ Synchronize
13:             else if $p2 = color$ then     ▷ Listen for a possible incoming $p1$th bit
14:                 LISTEN ; LISTEN          ▷ Then append the received bit in $mRec$
15:                 if beep heard in first round then $mRec[p3] := mRec[p3] \parallel 0$
16:                 if beep heard in second round then $mRec[p3] := mRec[p3] \parallel 1$
17:             else
18:                 LISTEN ; LISTEN                                  ▷ Synchronize
19: EndProtocol
```

The following lemma is straightforward.

Lemma 10. *Given a 2-hop c-coloring, the CONGEST model with edge bandwidth B can be simulated in the beeping model, with an $O(c^2 \cdot B)$ multiplicative factor.*

Finally, using the simulation of CONGEST, one can use the result of Barenboim and Elkin [16] (given for CONGEST), to obtain an $O(a)$-coloring in the beeping model. It is done by first computing, in the beeping model, a 2-hop (Δ^2+1)-coloring in $O(a^2\Delta^2\log^2 n + a^3\Delta^3\log n)$ rounds (Corollary 2). Then the $O(a)$-coloring from [16] (with $O(a^\mu \log n)$ round complexity) is combined with the CONGEST simulation, using the (Δ^2+1)-coloring obtained before. By Lemma 10, the resulting simulation of the $O(a)$-coloring protocol has $O(a^\mu\Delta^4\log^2 n)$ round complexity.

The final result is an $O((a^2\Delta^2 + a^\mu\Delta^4)\cdot\log^2 n + a^3\Delta^3\log n)$ time $O(a)$-coloring protocol in the beeping model. Notice that now by using this coloring algorithm, together with the $SquareSim$ protocol, to obtain a 2-hop $O(a\cdot\Delta)$-coloring (see Sect. V), we reduce the time multiplicative factor when simulating CONGEST algorithms. Consequently, one obtains a more efficient simulation.

References

[1] A. Cornejo and F. Kuhn, "Deploying wireless networks with beeps," in *DISC*, 2010, pp. 148–162.

[2] S. Navlakha and Z. Bar-Joseph, "Distributed information processing in biological and computational systems," *Commun. ACM*, vol. 58, no. 1, pp. 94–102, Dec. 2014.

[3] R. Guerraoui and A. Maurer, "Byzantine fireflies," in *DISC*, 2015, pp. 47–59.

[4] Y. Afek, N. Alon, Z. Bar-Joseph, A. Cornejo, B. Haeupler, and F. Kuhn, "Beeping a maximal independent set," *Distributed Computing*, vol. 26, no. 4, pp. 195–208, Aug 2013.

[5] D. Eppstein and D. Strash, "Listing all maximal cliques in large sparse real-world graphs," in *SEA*, 2011, pp. 364–375.

[6] C. S. A. Nash-Williams, "Decomposition of finite graphs into forests," *Journal of the London Mathematical Society*, vol. 39, p. 12, 1964.

[7] A. Casteigts, Y. Métivier, J. M. Robson, and A. Zemmari, "Design Patterns in Beeping Algorithms: Examples, Emulation, and Analysis," *ArXiv e-prints*, Jul. 2016.

[8] A. Scott, P. Jeavons, and L. Xu, "Feedback from nature: An optimal distributed algorithm for maximal independent set selection," in *PODC*, 2013, pp. 147–156.

[9] A. Casteigts, Y. Métivier, J. M. Robson, and A. Zemmari, "Design Patterns in Beeping Algorithms," in *OPODIS*, 2016, pp. 15:1–15:16.

[10] J. Schneider and R. Wattenhofer, "What is the use of collision detection (in wireless networks)?" in *DISC*, 2010, pp. 133–147.

[11] D. Peleg, *Distributed Computing: A Locality-Sensitive Approach*, ser. Monographs on Discrete Mathematics and Applications. Society for Industrial and Applied Mathematics, 2000.

[12] K.-T. Förster, J. Seidel, and R. Wattenhofer, "Deterministic leader election in multi-hop beeping networks," in *DISC*, 2014, pp. 212–226.

[13] B. Awerbuch, M. Luby, A. V. Goldberg, and S. A. Plotkin, "Network decomposition and locality in distributed computation," in *FOCS*, 1989, pp. 364–369.

[14] L. Barenboim and M. Elkin, *Distributed Graph Coloring: Fundamentals and Recent Developments*. Morgan & Claypool Publishers, 2013.

[15] A. J. Hoffman, K. Jenkins, and T. Roughgarden, "On a game in directed graphs," *Inf. Process. Lett.*, vol. 83, no. 1, pp. 13–16, 2002.

[16] L. Barenboim and M. Elkin, "Deterministic distributed vertex coloring in polylogarithmic time," in *PODC*, 2010, pp. 410–419.

[17] J. Schneider and R. Wattenhofer, "Distributed coloring depending on the chromatic number or the neighborhood growth," in *SIROCCO*, 2011, pp. 246–257.

[18] J. Schneider, M. Elkin, and R. Wattenhofer, "Symmetry breaking depending on the chromatic number or the neighborhood growth," *Theor. Comput. Sci.*, vol. 509, no. C, pp. 40–50, Oct. 2013.

[19] A. Casteigts, Y. Métivier, J. Robson, and A. Zemmari, "Deterministic leader election in $\mathcal{O}(D + \log n)$ time with messages of size $\mathcal{O}(1)$," in *DISC*, 2016, pp. 16–28.

[20] R. Cole and U. Vishkin, "Deterministic coin tossing with applications to optimal parallel list ranking," *Information and Control*, vol. 70, no. 1, pp. 32 – 53, 1986.

[21] L. Barenboim and M. Elkin, "Distributed (δ+1)-coloring in linear (in δ) time," in *STOC*, 2009, pp. 111–120.

[22] F. Kuhn, "Weak graph colorings: Distributed algorithms and applications," in *SPAA*, 2009, pp. 138–144.

[23] A. Goldberg, S. Plotkin, and G. Shannon, "Parallel symmetry-breaking in sparse graphs," in *STOC*, 1987, pp. 315–324.

[24] L. Barenboim and M. Elkin, "Sublogarithmic distributed mis algorithm for sparse graphs using nash-williams decomposition," in *PODC*, 2008, pp. 363–379.

[25] G. Agnarsson and M. Halldorsson, "Coloring powers of planar graphs," *SIAM Journal on Discrete Mathematics*, vol. 16, no. 4, pp. 651–662, 2003.

Appendix A
EBET

We remind that the "Balanced Execution Technique" (BET) from Förster *et al.* [12] guarantees $\mathcal{L}_1$-synchronization points. Now, we present an extension of BET, with which we guarantee $\mathcal{L}_i$-synchronization points for all $i \geq 1$. The "Extended Balanced Execution Technique" (EBET) allows the design of complex uniform protocols in the beeping model.

A. Introducing EBET

Synchronization points are not a natural primitive in the beeping model: an $\mathcal{L}_i$-synchronization point forces nodes which have reached a terminal $\mathcal{L}_i$-phase state (ended the $\mathcal{L}_i$-phase) to wait for their neighboring nodes to end the $\mathcal{L}_i$-phase, before starting the next one. Some protocols are difficult to design in a uniform manner without the use of synchronization points. Therefore, we want to be able to design a protocol $\mathcal{P}$ using synchronization points, and then apply a "technique" on the formal description of $\mathcal{P}$, so that the result is a protocol that can be run in the beeping model (not necessarily a formal description). The resulting protocol is called $\mathcal{P}_{sim}$. The technique we use for that is EBET.

EBET has two crucial components and a parameter k ($\in \mathbb{N}^{>0}$), which controls the small multiplicative overhead of EBET. The *first* component is a *Finite State Machine* (FSM), used to stall nodes when they have ended an $\mathcal{L}_i$-phase (*synchronization property*), for all $i \leq k$, so that other nodes can catch up (resulting in a resynchronization process for the start of the next $\mathcal{L}_i$-phase). The *second* is a *balanced round counter* rC, which is used so that nodes can reach some agreement on the clock value for the current $\mathcal{L}_1$-phase. By *balanced* counter, we mean that the rC values of two neighbors differ by at most 1 (*balancing property*). Thus two neighbors participating in the same $\mathcal{L}_1$-phase are in the same round, or in consecutive rounds. EBET's main addition is an extention to the FSM component. As a consequence, EBET provides $\mathcal{L}_i$-synchronization points, for all $i \leq k$. For better clarity, we consider EBET with $k = 2$, but it is simple to extend the following techniques for any given positive integer k.

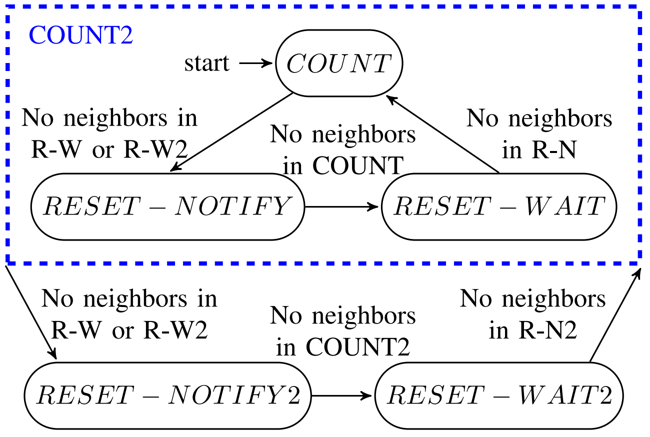
We assume that in $\mathcal{P}$ and $\mathcal{P}_{sim}$, all nodes start synchronously. By using synchronization points, $\mathcal{P}$ is easily described, coded and understood. Here we consider $\mathcal{P}$ to be a uniform loop of $\mathcal{L}_2$-phases (thus a uniform $\mathcal{L}_3$-phase). Whereas $\mathcal{P}_{sim}$ is a uniform loop of $\mathcal{L}_1$-phases, and each of its $\mathcal{L}_1$-phase simulates a round of $\mathcal{P}$. Since the $\mathcal{L}_1$-phases of $\mathcal{P}_{sim}$ contain exactly 11 rounds (referred to as *slots* to differentiate from the *rounds* in $\mathcal{P}$), $\mathcal{P}_{sim}$ can be run in the beeping model. We refer to phases of $\mathcal{P}$ as *original phases*, and to phases of $\mathcal{P}_{sim}$ as *simulation phases*. It is crucial that $\mathcal{P}_{sim}$ outputs the same result as $\mathcal{P}$, and proving this is the main focus of Sect. A-C.

In the first section, we describe the balanced counter technique (extending that of [12]), which allows EBET to maintain a balanced round counter, and to guarantee the synchronization property for all $\mathcal{L}_i$-phases. When abstracted to a higher level, the synchronization property results in the simulation of $\mathcal{L}_i$-synchronization points. In the second section, we describe how communication is adapted for EBET. Indeed, nodes do not have perfectly synchronized round counters, so we adapt the manner in which nodes communicate between themselves (having a balanced counter is crucial here), so as to simulate an execution with synchronized round counters.

B. Extending the Balanced Counter Technique for EBET

1) Slot Behavior in the Balanced Counter Technique: The balanced counter technique is implemented in the following manner. Nodes have the following variables: $state$, rC, p_1 and p_2. Theses variables are parametrized by a node v and if unclear, by a simulation $\mathcal{L}_1$-phase p, to indicate their value for v at the start of a simulation $\mathcal{L}_1$-phase p. The $state$ variable can be any of the 5 states from Figure 5 (CT, R-N, R-W, R-$N2$ and R-$W2$). A node v has $state(v,p) = CT$, it is said to be *participating* (in phase p), because it is simulating a round of an original $\mathcal{L}_1$-phase in $\mathcal{P}$. We define $CT2 = \{CT, R\text{-}N, R\text{-}W\}$, a composite state, and similarly, a node v with $state(v) \in CT2$ is simulating an original $\mathcal{L}_2$-phase in $\mathcal{P}$ (not necessarily a round).

Fig. 5: Finite State Machine Component for EBET ($k = 2$)



Each simulation $\mathcal{L}_1$-phase contains exactly 8 slots and is used to transmit a node's local clock value and its FSM state. Using this information, nodes know if they are ahead or behind of their neighbors, and act accordingly. The first three slots (indexed 0 to 2) are used to transmit the counter value (rC) modulo 3, and the other slots (indexed 3 to 7) are used to transmit the current FSM state of a node ($state$). For any given node v, the information is transmitted in the following manner during each simulation $\mathcal{L}_1$-phase:

- If $state = CT$, then v beeps in slots (rC mod 3) and 3,
- If $state = R$-N, then v beeps in slot 4,
- If $state = R$-W, then v beeps in slots rC mod 3 and 5,
- If $state = R$-$N2$, then v beeps in slot 6,
- If $state = R$-$W2$, then v beeps in slots rC mod 3 and 7.

Node v listens in all slots it does not beep in.

Now, we describe the state transitions of the FSM, and their guard conditions (also shown in Figure 5). These conditions are essential for the *balanced counter* and *synchronization* properties (Lemmas 13 and 14) in EBET. For any given node v, the allowed state transitions are:

1) $CT \to R$-N if no node $u \in \mathcal{N}(v)$ is in R-W or R-$W2$,
2) R-$N \to R$-W if no node $u \in \mathcal{N}(v)$ is in CT,
3) R-$W \to CT$ if no node $u \in \mathcal{N}(v)$ is in R-N,
4) $CT \to R$-$N2$ if no node $u \in \mathcal{N}(v)$ is in R-W or R-$W2$,
5) R-$N2 \to R$-$W2$ if no node $u \in \mathcal{N}(v)$ is in $CT2$,
6) R-$W2 \to CT$ if no node $u \in \mathcal{N}(v)$ is in R-$N2$.

The state transitions of the FSM can be decomposed into two cycles. We denote the first cycle (transitions $1 \to 2 \to 3$) as an $\mathcal{L}_1$-*cycle*, and the second cycle (transitions $4 \to 5 \to 6$) as an $\mathcal{L}_2$-*cycle*. An $\mathcal{L}_1$-cycle is used to transition to the next original $\mathcal{L}_1$-phase (if there is one) of the current original $\mathcal{L}_2$-phase being simulated (from $\mathcal{P}$), and essentially implements an $\mathcal{L}_1$-synchronization point. Similarly, an $\mathcal{L}_2$-cycle is used to transition to the next original $\mathcal{L}_2$-phase (if there is one) in $\mathcal{P}$, and essentially implements an $\mathcal{L}_2$-synchronization point.

In terms of states, an $\mathcal{L}_1$-cycle goes $CT \to R$-$N \to R$-$W \to CT$. R-N is used to indicate the executing node has finished the simulated original $\mathcal{L}_1$-phase. Nodes in that state do not interfere with their neighbors' simulations of that original $\mathcal{L}_1$-phase, as the balanced counter rC is not transmitted: no beeps in the first three slots.
On the other hand, R-W is used to indicate the node is starting the next original $\mathcal{L}_1$-phase. Nodes in that state stall neighboring nodes participating in that next original $\mathcal{L}_1$-phase in two different ways. First, a rC value of 0 is transmitted, which stalls the $increment$ function of these neighboring nodes (see following section, Sect. A-B2). As such, the neighboring rC values satisfy $rC \leq 1$ (Lemma 12). Second, neighboring nodes are prevented from transitioning to R-N or R-$N2$ until the node participates, i.e., transitions to CT (see the conditions of transitions 1 and 4).
Since R-W interferes with participating nodes while R-N does not, the synchronization property relies heavily on the conditions of transition 2. That is, a node remains in R-N while its neighbors simulate additional rounds of the original $\mathcal{L}_1$-phase, and only transitions once all neighboring nodes have finished, i.e., transitioned to R-N.

In the same way, an $\mathcal{L}_2$-cycle goes $CT \to R$-$N2 \to R$-$W2 \to CT$. The R-$N2$ (resp., R-$W2$) state acts similarly to the R-N (resp., R-W) state. Since R-$W2$ interferes with participating nodes, as well as nodes going through a $\mathcal{L}_1$-cycle, while R-$N2$ does not, the synchronization property relies heavily on the conditions of transition 5. That is, a node remains in R-$N2$ while its neighbors simulate additional $\mathcal{L}_1$-phases of the original $\mathcal{L}_2$-phase (either participating or going through $\mathcal{L}_1$-cycles), and only transitions to the next $\mathcal{L}_2$-phase once all neighbors have finished, i.e., transitioned to R-$N2$.

The synchronization property results from the following observations. Two neighboring nodes going through a $\mathcal{L}_1$-cycle (resp. $\mathcal{L}_2$-cycle) are always in two consecutive states of the $\mathcal{L}_1$-cycle (resp., $\mathcal{L}_2$-cycle), due to the transition conditions. In other words, they have gone through the same number of $\mathcal{L}_1$-cycles (resp., $\mathcal{L}_2$-cycles), unless a node participates and its neighbors is still in state R-W (resp., R-$W2$). In which case, the participating node can neither increment its balanced counter (beyond 1), nor transition to any other state, and thus waits for its neighbor. Finally, consider two neighboring nodes, one going through a $\mathcal{L}_1$-cycle and the other through a $\mathcal{L}_2$-cycle. Then the second node's state is necessarily R-$N2$, and it neither interferes with the first node, nor transitions before the first node enters a $\mathcal{L}_2$-cycle (i.e., enters the $R\text{-}N2$ state).

2) Functions of the Balanced Counter Technique: The $\mathcal{L}_1$ and $\mathcal{L}_2$-cycles, as well as the balanced counter rC, are managed by the following functions: $reset$, $reset2$ and $increment$. These functions can only be invoked by participating nodes and increment $p1$, $p2$ and rC while ensuring the synchronization and balancing properties. When node v increments $p1$ (resp., $p2$), that means that v has done a full $\mathcal{L}_1$-cycle (resp., $\mathcal{L}_2$-cycle). Consequently, $p1$ (resp., $p2$) counts the number of $\mathcal{L}_1$-synchronization points invoked in the current original $\mathcal{L}_2$-phase (resp., the number of $\mathcal{L}_2$-synchronization points invoked). The *synchronization property*, which states that $p1$ and $p2$ are the same for two neighboring participating nodes, means that they are simulating the same original $\mathcal{L}_1$-phase.

We define a boolean $next(v,p)$, used in the following function, for any given simulation $\mathcal{L}_1$-phase p and node v. The boolean is true if and only if all neighboring nodes of v have equal or greater rC values. v learns its $next$ value after the first three slots of p, since the boolean is true if and only if v detects no beeps in slot $rC(v) - 1$ mod 3. If $next(v,p)$ is true, then $rC(v)$ is incremented at the end of phase p.

$increment$ is used to increment rC without violating the *balancing property*. Node v calls $increment$ in the very first phase of $\mathcal{P}_{sim}$ (and whenever an original $\mathcal{L}_1$-phase starts), and calls $increment$ again whenever the previous call finishes, until the original $\mathcal{L}_1$-phase is finished. During these calls, v simulates $\mathcal{P}$ since $state(v) = CT$. When $increment$ is invoked by a node v, v waits for the first simulation $\mathcal{L}_1$-phase p in which $next(v,p)$ is true. At the end of this phase, v increments rC.

$reset$ is used to go through a full $\mathcal{L}_1$-cycle. When invoked by v, v goes through a full $\mathcal{L}_1$-cycle (transitions 1, 2 and 3). During the cycle, $rC(v)$ is reset to 0 after transition 1 succeeds and $p_1(v)$ is incremented after transition 3 succeeds. Similarly, $reset2$ is used to go through a full $\mathcal{L}_2$-cycle (transitions 4, 5 and 6). During the cycle, $rC(v)$ and $p_1(v)$ are reset to 0 after transition 4 succeeds and $p_2(v)$ is incremented after transition 6 succeeds. The $reset$ (resp. $reset2$) function simulates a $\mathcal{L}_1$-synchronization point (resp. $\mathcal{L}_2$-synchronization point): it is invoked by a participating node v in the round after it reaches a $\mathcal{L}_1$-synchronization point (resp. $\mathcal{L}_2$-synchronization point), when simulating $\mathcal{P}$. The details are in Sect. A-C.

3) Properties of the Balanced Counter Technique: First, we give a few lemmas (Lemmas 11 and 12), which are then used to prove both the balancing and synchronization properties (Lemmas 13 and 14).

Lemma 11. *For any given simulation $\mathcal{L}_1$-phase p and node v, if $state(v,p) = R\text{-}W2$, then for all $u \in \mathcal{N}(v)$ $state(u,p) \notin \{R\text{-}N, R\text{-}W\}$.*

Proof. Let us prove this lemma by induction on p. Trivially true for $p = 0$ because of the initialization conditions ($state(v,0) \neq R\text{-}W2$).
For the induction step, by contradiction, let us consider a node $u \in \mathcal{N}(v)$, such that $state(u,p) \in \{R\text{-}N, R\text{-}W\}$. Since at most one transition can be enacted by a node per phase, we know $state(v,p-1) \in \{R\text{-}N2, R\text{-}W2\}$ and $state(u,p-1) \in CT2$. It is not possible that $state(v,p-1) = R\text{-}N2$ because of the condition for transition 5. Now, consider $state(v,p-1) = R\text{-}W2$. It is not possible that $state(u,p-1) = CT$ because of the condition for transition 1, and it is not possible that $state(u,p-1) \in \{R\text{-}N, R\text{-}W\}$ due to the induction hypothesis. As a result, for all $u \in \mathcal{N}(v)$ $state(u,p) \notin \{R\text{-}N, R\text{-}W\}$. □

Lemma 11 is used to simplify the proof of Lemma 12. It also highlights the fact that nodes which have ended the current $\mathcal{L}_2$-phase are stalled in the $R\text{-}N2$ state until all of their neighbors also end the $\mathcal{L}_2$-phase.

Lemma 12. *For any given simulation $\mathcal{L}_1$-phase p and node v, if $state(v,p) \in \{R\text{-}W, R\text{-}W2\}$, then for all $u \in \mathcal{N}(v)$ $rC(u,p) \leq 1$.*

Proof. Let us prove this lemma by induction on p. Trivially true for $p = 0$ because of the initialization conditions ($state(v,0) \notin \{R\text{-}W, R\text{-}W2\}$).
For the induction step, consider a given simulation $\mathcal{L}_1$-phase p and node v, where $state(v,p) \in \{R\text{-}W, R\text{-}W2\}$. Since $state(v,p) \neq CT$, $rC(v,p) = 0$. Consider any given neighboring node u of v. If $state(u,p) \neq CT$, then $rC(u,p) = 0$. Now, consider $state(u,p) = CT$. Let us prove that $rC(u,p) \leq 1$.

First, consider $state(v,p) = R\text{-}W$. Since at most one transition can be enacted by a node per phase, we know $state(v,p-1) \in \{R\text{-}N, R\text{-}W\}$ and $state(u,p-1) \in \{CT, R\text{-}W, R\text{-}W2\}$. By Lemma 11, $state(u,p-1) \neq R\text{-}W2$. It is also not possible that $state(v,p-1) = R\text{-}N$. The condition for transition 2 renders $state(u,p-1) = CT$ impossible, and $state(u,p-1) = R\text{-}W$ is also impossible, because u is then unable to enact transition 3 at the same time that v enacts transition 2. Finally, the remaining possibilities are that $state(v,p-1) = R\text{-}W$ and $state(u,p-1) \in \{CT, R\text{-}W\}$. For $state(u,p-1) = R\text{-}W$, since u enacts transition 3 at the end of phase $p-1$, $rC(u,p) = 0$. As for $state(u,p-1) = CT$, $rC(u,p-1) \leq 1$ by induction hypothesis. Since v stalls u (state $R\text{-}W$ and $rC(v,p-1) = 0$), u is unable to increment rC higher than 1 at the end of phase $p-1$ and $rC(u,p) \leq 1$.

Now, consider $state(v,p) = R\text{-}W2$. Since at most one transition can be enacted by a node per phase, we know $state(v,p-1) \in \{R\text{-}N2, R\text{-}W2\}$ and $state(u,p-1) \in \{CT, R\text{-}W, R\text{-}W2\}$. First, consider $state(v,p-1) = R\text{-}N2$. Since at the end of phase $p-1$, v enacts transition 5, we have $state(u,p-1) \notin CT2$. However, it is also impossible that $state(u,p-1) = R\text{-}W2$, because then at the end of phase $p-1$, u and v enacts respectively transition 6 and 5. Now, let us consider $state(v,p-1) = R\text{-}W2$. By Lemma 11, it is impossible that $state(u,p-1) = R\text{-}W$. If $state(u,p-1) = R\text{-}W2$, then as u enacts transition 6 at the end of phase $p-1$, $rC(u,p) = 0$. Otherwise, if $state(u,p-1) = CT$, then by induction hypothesis, $rC(u,p-1) \leq 1$. Since v stalls u, $rC(u,p) \leq 1$. □

Using Lemma 12, which states that nodes in $R\text{-}W$ or in $R\text{-}W2$ stall the balanced counters of neighboring participating nodes, we prove the balancing property (Lemma 13).

Lemma 13 (Balancing property)**.** *For any given simulation $\mathcal{L}_1$-phase p and two neighboring participating nodes u and v, $|rC(v,p) - rC(u,p)| \leq 1$.*

Proof. Let us prove that rC satisfies the balancing property by induction on p. For $p = 0$, the balancing property is given by the initialization conditions.
For the induction step, consider a simulation $\mathcal{L}_1$-phase $p > 0$ and two neighboring nodes u and v. In the first case, u and v were participating in simulation $\mathcal{L}_1$-phase $p-1$. Then by the induction hypothesis for $p-1$, $|rC(u,p-1) - rC(v,p-1)| \leq 1$. Since counters can only increase by one per simulation $\mathcal{L}_1$-phase, and $increment$ stalls nodes which are ahead, the induction hypothesis holds. In the second case, at least one of the nodes was not participating in simulation $\mathcal{L}_1$-phase $p-1$. W.l.o.g, u was not participating. Due to the transition restrictions, u was in $R\text{-}W$ or $R\text{-}W2$ in $p-1$ (node u cannot transition from $R\text{-}N$ to CT in a single phase). Thus, by Lemma 12, $rC(v,p-1) \leq 1$. The same line of arguments as above shows that the induction hypothesis holds. □

The balancing property highlights the fact that early nodes are stalled by neighboring nodes with smaller counters, and so on until the latest node. However this latest node is never stalled, and thus controls the increment rate of all balanced counters. Once this node catches up with the other nodes, all nodes increment their round counters synchronously. Thus, from the perspective of the latest node, the balanced counters are fully synchronized counters.

Now, we prove the synchronization property (Lemma 14), which states that p_1 (resp., p_2) is an index for original $\mathcal{L}_1$-phases (resp., $\mathcal{L}_2$-phases). As a result, two neighboring participating nodes are simulating the same $\mathcal{L}_1$-phase (in the same $\mathcal{L}_2$-phase).

Lemma 14 (Synchronization property)**.** *For any given simulation $\mathcal{L}_1$-phase p and two neighboring nodes u and v, if $state(u,p) = state(v,p) = CT$ then $p_1(v,p) = p_1(u,p)$ and $p_2(v,p) = p_2(u,p)$. It can also be said that u and v have invoked $reset2$ the same number of times, and have invoked $reset$ the same number of times since they last invoked $reset2$.*

Proof. Let us prove by induction on p, that for any given simulation $\mathcal{L}_1$-phase p and two neighboring nodes u and v:

1) if $state(u,p) = R\text{-}W$ and $state(v,p) = CT$ (or vice versa) then $p_1(v,p) = p_1(u,p) + 1$ and $p_2(v,p) = p_2(u,p)$,
2) else if $state(u,p) = R\text{-}W2$ and $state(v,p) = CT$ (or vice versa) then $p_1(v,p) = p_1(u,p) = 0$ and $p_2(v,p) = p_2(u,p) + 1$,
3) else if $state(u,p) = CT2$ and $state(v,p) = R\text{-}N2$ (or vice versa) then $p_1(v,p) = 0$ and $p_2(v,p) = p_2(u,p)$,
4) otherwise, $p_1(v,p) = p_1(u,p)$ and $p_2(v,p) = p_2(u,p)$.

The synchronization property corresponds to the case when $state(u,p) = state(v,p) = CT$.
Trivially true for $p = 0$ because of the initialization conditions.
For the induction step, consider a simulation $\mathcal{L}_1$-phase $p > 0$ and two neighboring participating nodes u and v.

First, consider $state(u,p) = R\text{-}W$ and $state(v,p) = CT$. By lemma 11, $state(v,p-1) \neq R\text{-}W2$. Because it is not possible for both u and v to transition at the end of phase $p-1$ (see condition for transition 3), or for u to transition from $R\text{-}N$ to $R\text{-}W$ if v stays in CT (see condition for transition 2), $state(u,p-1) \neq R\text{-}N$. Thus, consider $state(u,p-1) = R\text{-}W$. We know $state(v,p-1) = CT$ or $state(v,p-1) = R\text{-}W$. Thus, by induction hypothesis (items 1 and 3) for $p-1$, item 1 of the induction hypothesis holds.

Now, consider $state(u,p) = R\text{-}W2$ and $state(v,p) = CT$. Suppose by contradiction that $state(u,p-1) = R\text{-}N2$. Because of the condition for transition 5, the only possibility is that $state(v,p-1) = R\text{-}W2$. However, it is impossible for both u and v to transition at the end of phase $p-1$ (see condition for transition 6). Thus, consider $state(u,p-1) = R\text{-}W2$. By lemma 11, $state(v,p-1) \neq R\text{-}W$. Thus, either $state(v,p-1) = R\text{-}W2$, or $state(v,p-1) = CT$. And by induction hypothesis (items 2 and 3) for $p-1$, item 2 of the induction hypothesis holds.

Then, consider $state(u,p) = CT2$ and $state(v,p) = R\text{-}N2$. Since at most one transition can be enacted by a node per phase, we know $state(u,p-1) \in CT2 \cup \{R\text{-}W2\}$ and $state(v,p-1) \in \{CT, R\text{-}N2\}$. First, consider $state(v,p-1) = R\text{-}N2$. It is impossible that $state(u,p-1) = R\text{-}W2$, because of the condition for transition 6. Then, $state(u,p-1) \in CT2$ and we can rely on the induction hypothesis (item 3) for $p-1$. Now, consider $state(v,p-1) = CT$. Because of the condition for transition 4, it is impossible that $state(u,p-1) = R\text{-}W2$. Thus, by induction hypothesis (items 1 and 4) for $p-1$ and the definition of $reset2$ ($p1$ is reset after transition 4 succeeds), item 3 of the induction hypothesis holds.

Finally, consider the other cases. Then either $state(u,p) = state(v,p)$, or $state(u,p) \neq state(v,p)$. Moreover, for $p-1$, then either $state(u,p-1) = state(v,p-1)$ or $state(u,p-1) \neq state(v,p-1)$. By using the induction hypothesis (all items), item 4 of the induction hypothesis holds. □

Using the balancing and synchronization properties, we can simulate fully synchronized round counters (as in BET) with rC. Consequently, EBET simulates the rounds of an original $\mathcal{L}_1$-phase.

C. Balanced Executions in EBET

We extend the simulation $\mathcal{L}_1$-phases with 3 additional slots. Thus, a simulation $\mathcal{L}_1$-phase contains 11 slots. The 3 extra slots are dedicated to the simulation of a round r in $\mathcal{P}$. That simulated round is either rC or $rC - 1$, depending on the rC values of the neighboring nodes.
We define a *correct action*, for any given participating node v and simulation $\mathcal{L}_1$-phase p of $\mathcal{P}_{sim}$. v's action when simulating round r in simulation $\mathcal{L}_1$-phase p is said to be *correct* if it is the same as v's action in round r of $\mathcal{P}$. We prove that all actions (simulating rounds of $\mathcal{P}$) done by nodes in $\mathcal{P}_{sim}$ are correct. Thus, $\mathcal{P}_{sim}$ and $\mathcal{P}$ have the same result.

1) Rules to ensure Balanced Execution: We give the following additional rules. They ensure, that for any given participating node v and simulation $\mathcal{L}_1$-phase p of $\mathcal{P}_{sim}$, v's actions in $\mathcal{L}_1$-phase p is correct.

- If $next(v,p) = false$, v simulates round $rC(v,p) - 1$.
- Otherwise, v simulates round $rC(v,p)$.

A round r is simulated by v in the following way. If v's action for r is $BEEP$, then v beeps in slot r mod 3 + 8 of simulation $\mathcal{L}_1$-phase p, and otherwise it listens in that slot.

With the rules above, the following definitions are natural. For any given node v and for any simulated round r of $\mathcal{P}$, we define $p_n(v,r)$ as the first simulation $\mathcal{L}_1$-phase p in which v simulates the next round $(r+1)$. We also define $p_f(v,r)$ as the first simulation $\mathcal{L}_1$-phase p in which v simulates round r.

Now, consider *end of phase rounds* of $\mathcal{P}$ (rounds in which a node ends an $\mathcal{L}_i$-phase and thus reaches a $\mathcal{L}_i$-synchronization point). A participating node v detects whether it reaches a $\mathcal{L}_i$-synchronization point after round r of $\mathcal{P}$ in simulation $\mathcal{L}_1$-phase $p_n(v,r)$, since in that phase, v is already done with beeping or listening to beeps for round r (as even the slowest neighbors simulated r in the previous phase). Consequently, consider rF as the round after which v reaches a $\mathcal{L}_i$-synchronization point in $\mathcal{P}$. v invokes $reset$ or $reset2$ (depending on the synchronization point) in simulation $\mathcal{L}_1$-phase $p_n(v,rF)$, which ensures the simulation of $\mathcal{P}$ is correct.

2) Simulation Proofs: First, we give the following simple lemma. It states that when a node v is simulating round $rC(v,p)-1$ in a simulation $\mathcal{L}_1$-phase p, it has already simulated the round once, in a previous simulation phase. The round is simulated again while v is waiting for the slower nodes (with smaller rC values), until $next(v)$ is true, in which case all neighboring nodes have caught up.

Lemma 15. *For any given phase $p > 0$ and participating node v, v has already simulated round $rC(v) - 1$ at least once.*

Now, we prove a crucial lemma (Lemma 16). Basically, it states that for any simulation $\mathcal{L}_1$-phase p, all nodes have correctly simulated $\mathcal{P}$ for all rounds $r < rC(v,p)$. Moreover, in the round in which a participating node v increments $rC(v)$, $rC(v)$ is simulated correctly, due to the fact that all neighbors have already acted once for $rC(v) - 1$, and that all of these actions were correct. Using this lemma, obtaining Theorem 9 is straightforward.

Lemma 16. *For any given simulation $\mathcal{L}_1$-phase p and participating node v, all previous actions from v were correct.*

1) Moreover, if $next(v,p) = true$:
  a) If v listens for round $rC(v) - 1$: $\exists u \in \mathcal{N}(v)$, u participating, s.t. u beeps for $rC(v) - 1 \Leftrightarrow v$ detects a (correct) beep for $rC(v) - 1$ in a phase $p' < p$.
  b) If v beeps for round $rC(v) - 1$: $\exists u \in \mathcal{N}(v)$, u participating, s.t. u listens for $rC(v) - 1 \Leftrightarrow u$ detects a (correct) beep for $rC(v) - 1$ in a phase $p' < p$.
  c) v's action for round $rC(v,p)$ is correct,

2) Otherwise, v's action for round $rC(v,p) - 1$ is correct.

Proof. Let us prove this lemma by induction on the simulation $\mathcal{L}_1$-phase p. For $p = 0$, the induction hypothesis (IH) holds obviously.

For the induction step, consider a phase $p > 0$ and any given participating node v. First, from the IH in phase $p-1$, we get that all actions done by v previous to phase $p-1$ were correct, as well as the action v executed in $p-1$.

Next, let us prove part 1a and 1b of the IH. Consider any given phase p' in which v or any of its neighbors simulates $rC(v,p) - 1$ for the first time. In part 1 of the IH, $next(v,p) = true$ thus $p' < p$ (Lemma 15).
Let us prove ($\Rightarrow$) of parts 1a and 1b. Consider $u \in \mathcal{N}(v)$ s.t. u beeps (resp. listens) for $rC(v,p) - 1$. We prove v detects u's beep (resp. u detects v's beep). The faster node of the pair (u and v) is stalled by the slower node. When the slower node first simulates $rC(v,p) - 1$ in a phase $p' < p$, the faster node w is still simulating $rC(v,p) - 1$ because $next(w,p')$ is false. Thus, in p', v detects u's beep (resp. u detects v's beep). By the IH, any beep heard is correct.
($\Leftarrow$) follows from the fact that beeps are transmitted to neighboring nodes only and because of the manner in which the last three slots are used (and non participating nodes do not use them).
Since all previous actions done by v were correct and part 1a of the IH holds (for phase p), part 1c of the IH holds.

Finally, let us prove part 2 of the IH. Suppose $next(v,p) = false$. We know v's action for $rC(v,p) - 1$ in phase $p-1$ is correct, by part 1a of the IH or part 2 of the IH (depending on $next(v,p-1)$). Since the action chosen by v for round $rC(v,p) - 1$ does not change, part 2 of the IH holds. □

Theorem 9. *The outputs of $\mathcal{P}$ and $\mathcal{P}_{sim}$ are identical.*

Finally, let us prove that the round complexity of $\mathcal{P}_{sim}$, R_{sim}, is close to R, the round complexity of $\mathcal{P}$. Theorem 10 states that using EBET impacts the round complexity by a small multiplicative factor only. It should be noted though, that R is dependent on the $\mathcal{L}_i$-synchronization points. Indeed, $\mathcal{P}$ might be slowed by the synchronization points (due to the "global" resynchronization process). However, when each original $\mathcal{L}_i$-phase's round complexity is bounded independently of a parameter (in particular, the diameter D), R is independent of that parameter, and R_{sim} is also independent of that parameter.

Theorem 10. *Let R_{sim} be the round complexity of $\mathcal{P}_{sim}$ and R be that of $\mathcal{P}$. Then $R_{sim} = O(R)$.*

Proof. First, there is a constant factor (here 11) between the number of rounds and the number of simulation $\mathcal{L}_1$-phases in $\mathcal{P}_{sim}$. Thus, we compare the number of simulation $\mathcal{L}_1$-phases in $\mathcal{P}_{sim}$ and the number of rounds in $\mathcal{P}$.
Let L be any given original $\mathcal{L}_1$-phase of $\mathcal{P}$. Let w be the node which takes the most rounds to end L, that quantity being r_w. In $\mathcal{P}_{sim}$, for any given node, at most r_w simulation $\mathcal{L}_1$-phases are used to simulate L. This holds for all original $\mathcal{L}_1$-phases, and starting the next $\mathcal{L}_1$-phase takes a constant number of simulation $\mathcal{L}_1$-phases. Moreover, $\mathcal{P}$ uses $\mathcal{L}_i$-synchronization points at the end of every original $\mathcal{L}_i$-phase, thus its round complexity R is the sum of the round complexities of all original $\mathcal{L}_1$-phases. Consequently, we have $R_{sim} = O(R)$. □