Overcoming interference in the beeping communication model

Fabien Dufoulon

université PARIS-SACLAY

UNIVERSITÉ PARIS SUD
Comprendre le monde, construire l'avenir

NNT : 2019SACLS233

Thèse de doctorat

Overcoming interference in the beeping communication model

Thèse de doctorat de l'Université Paris-Saclay
préparée à Université Paris-Sud

Ecole doctorale n°580 Sciences et technologies de l'information et de la communication (STIC)
Spécialité de doctorat : Informatique

Thèse présentée et soutenue à Orsay, le 27/09/2019, par

FABIEN DUFOULON

Composition du Jury :

Colette Johnen Professeur, Université de Bordeaux (LaBRI)	Président
Arnaud Casteigts Maître de Conférences (habilitation), Université de Bordeaux (LaBRI)	Rapporteur, absent
Roger Wattenhofer Professeur, ETH Zurich	Rapporteur
Pierre Fraigniaud Directeur de Recherche, Université Paris Diderot (IRIF)	Examinateur
Devan Sohier Professeur, Université de Versailles Saint-Quentin-en-Yvelines (Li-Parad)	Examinateur
Joffroy Beauquier Professeur émérite, Université Paris-Sud (LRI)	Directeur de thèse
Janna Burman Maître de Conférences, Université Paris-Sud (LRI)	Co-encadrante de thèse
Shay Kutten Professeur, Technion - Israel Institute of Technology (IE&M)	Invité

Acknowledgements

In writing a thesis one walks a difficult and harsh path. Numerous failures give birth to a bare handful of successes. These failures pull you down but in times of need, those around you help you up. For that reason, I am immensely grateful towards the people that have been by my side, no matter how briefly.

First of all, I would like to thank my advisors, Joffroy Beauquier and Janna Burman, for their guidance and support during these last three years. They taught me many things, from presenting my research ideas, to writing research papers and giving engaging talks about my results. In particular, I am incredibly grateful for their willingness to read my (unfortunately numerous) drafts and for taking considerable amount of their time to point out how I could improve. It is with fond memories and with bittersweet regret that I graduate and start new research projects.

I would like to express my gratitude to Roger Wattenhofer and Arnaud Casteigts for reviewing this document and for their feedback on my work. I am thankful to Colette Johnen, Pierre Fraigniaud and Devan Sohier for accepting to participate in my thesis committee.

I would also like to thank Shay Kutten and Yuval Emek for hosting me at the Technion twice during the course of my thesis, and Peter Davies for our co-authored paper and more generally for our discussions regarding the beeping model.

When I started engineering school, research was not something I had in mind. I am extremely grateful towards Joanna Tomasik, Arpad Rimmel and Johanne Cohen for helping me discover and enjoy research. Without them, I would have never known research or distributed computing, both of which are now an integral part of my life.

Also, I would like to warmly thank my colleagues (and friends) with whom I have spent time during these last three years. Beyond sharing our advisors, we have also spent a lot of great time together and for that, special thanks go to Marie Laveau, with whom I have shared an office for these last three years, and Chuan Xu, who has been incredibly supportive of me since I first arrived in the team. I am grateful to all of my colleagues in the Parsys team, and in particular to Thomas, Amal, Dajung, Oguz, Laertio, Timothée, Evangelos, Antoine and Ian. Outside the team, I am grateful to Pierre, Lou, Christian, David, Aygul, Houssem, Hai, Alexandre, Youfang and Jules for being good friends. Last but not least, I am happy to have spent time with Ami, Matthias, Laurent, Mikaël and Yackolley, and hope to meet them again, wherever our paths may lead.

Finally, I would like to thank everyone from the Laboratoire de Recherche en Informatique, the research staff as well as the technical and administrative staff, for being kind, helpful and overall incredibly nice people.

Contents

Chapter 1

Introduction

Small inexpensive inter-communicating electronic devices have become widely available. As a result, solutions for networks of such devices should be scalable and particularly resource efficient (in terms of energy consumption, time, exchanged data, etc.). The design and analysis of such solutions is one of the focuses of *distributed computing* - the study of distributed systems.

Electronic devices can differ greatly in capabilities depending on the application and cost restrictions. In particular, devices with severely limited capabilities (e.g., basic communication, constant-size memory or limited mobility), referred to as *weak devices*, have become more and more common. Systems of weak devices are at the heart of an ever-increasing amount of emerging high-potential applications, which include low energy transmission wireless networks, robot swarms, programmable matter and DNA-based computing devices. At a high level, these applications are core to highly anticipated innovative technologies such as advanced robotics, biological computing and the Internet of Things. Therefore, it is not surprising that distributed systems of weak devices have recently received much interest from the distributed computing community. Importantly, studying distributed systems of weak devices can give insight into resource-efficient design of distributed systems composed of stronger devices. Additionally, notice that it has become apparent that the multitude of electronic networks underpinning modern society consumes an increasingly large amount of energy. This is at odds with the global call for energy awareness. As a result, it is extremely important to understand how these networks can function in a more energy efficient manner.

Distributed systems lack a centralized control - a central authority with a full view of the system, assigning tasks to nodes accordingly. Instead, nodes take individual decisions in order to achieve some common goal (i.e., to solve a *distributed problem*). In order for these decisions to be correct and coordinated, nodes share information (e.g., by sending messages or writing on a shared memory) but within a certain limit since communication is resource-consuming. An active line of research consists in building efficient information transmission methods (i.e., *communication primitives*) upon a resource-limited but energy-efficient communication mechanism. Importantly, such communications [8, 74, 94, 38, 55] underpin low energy wireless (radio) networks.

Drawing additional interest to this work, parallels have been drawn between distributed systems with severely limited communication and some biological systems [85, 57, 86]. These include biological cellular systems, which communicate through protein secretions [37, 20, 2], as well as swarms of fireflies, which communicate through flashes of light [21]. Studying such systems may suggest solutions useful in the electronic world, since natural distributed systems are generally simple, robust

and resource-efficient.

1.1 Challenges of Limited Communication

Limited communication mechanisms generally suffer from some of the three following aspects. First, only a small amount of information can be transmitted in a single communication. Second, the sender may not be detectable a priori (e.g., information about the sender may not be included in the message). Third, simultaneous communications may result in information loss. This is common when communications use a shared medium (e.g., wireless communications or Ethernet). Such simultaneous communications are often called *collisions*.

Low-energy wireless networks build upon a severely limited communication mechanism that suffers from all three aspects. Nodes communicate using simple bursts of energy and *carrier sensing*: nodes detect the presence of energy bursts and cannot easily exchange information as e.g. in the conventional message-passing models. Recent research has focused on this simple communication [44, 43, 92, 82, 94, 38, 55] and how to communicate information efficiently. As a first step, efficient control of interference is crucial. However additional factors, such as the degree of synchronization between nodes, may impact the communication efficiency.

Interference Control. Multiple nodes in the same neighborhood (defined by an underlying ad-hoc structure of communication: the *communication graph*), communicating at the same time, produce some kind of *interference*. In traditional wireless networks, interference is *destructive*: messages sent by neighboring nodes are received only if there are no collisions, and their contents are lost otherwise (i.e., two or more messages cannot be distinguished from zero messages, but can be distinguished from a single message). With destructive interference, the information loss is extreme. As a result, information transmission often builds upon collision avoidance methods [10, 33], which are highly time-consuming.

On the other hand, a carrier-sensing-like communication generally can provide *non-destructive* interference: a node with two or more neighbors communicating simultaneously, detects that some communication happened rather than nothing. Thus, collisions are detected. They convey information, if only a very small amount. An important challenge lies in understanding how non-destructive interference can be controlled efficiently to allow for faster information transmission [92, 94].

Synchronization between Nodes. Some degree of synchronization between (neighboring) nodes' local clocks may be leveraged to obtain faster communication. If such synchronization is available, communication may take advantage of the clock values to convey additional information. On the other hand, without any synchronization, are nodes able to communicate information, and if yes, in a time-efficient manner?

It is necessary to better understand how much synchronization is required in low-energy wireless networks, and the trade-offs between synchronization and efficient communication in such networks.

1.2 The Beeping Model

Distributed systems composed of *weak devices with severely limited communication* capabilities - using simple bursts of energy - are the main focus of this thesis. A burst of energy can be thought of as a basic unmodulated wireless communication, a simple flash of light [64] or even protein secretion [2]. We are interested in using theoretical tools to analyze such systems. Towards that end, we rely on a formal communication model, which provides a formal foundation for the design and analysis of provably correct (using analytical tools) solutions.

The systems of our concern have been modeled by *the beeping communication model* [38]. Nodes wake up at some arbitrary times and then communicate in a synchronous manner (i.e., in *rounds*) using an extremely limited communication mechanism: nodes can either beep or listen in each round. The beeps are bursts of energy that bear no content - basically empty messages. As for listening nodes, they use carrier sensing. As such, they can only detect whether at least one of their neighbors (according to the communication graph) is beeping, or if none are. Consequently, collisions in the beeping model produce non-destructive interference. Notice that due to such collisions, a beep that originally transfers less than a bit - being a unary signal - may transfer even less information, when multiple beeps in the neighborhood are merged into one.

The beeping model differs from classical approaches in distributed computing. Indeed, beeps are not an abstract representation of a physically-complex message-passing communication. Traditionally, reliable communication in wireless networks requires an underlying multiple access method - also known as Medium Access Control (e.g., TDMA , FDMA or CDMA schemes) - to deal with multiple incoming messages. This underlying layer tends to be highly affected by the physical reality of wireless transmissions. Abstracting away this physical reality allows for clear and provably correct solutions at a possible cost to efficiency.

In contrast, beeps are a very simple low-level communication mechanism [17]. This however also has many advantages. First, as means of communication, beeps are well-behaved. A beep takes a short, well-defined amount of time and consumes very little energy. Although simultaneous beeps cause non-destructive interference, this type of interference is easier to deal with compared to destructive interference. Moreover, the beeping communication mechanism does not require many assumptions, which allows positive results in this model to be widely applicable to other distributed computing models. Finally, the beeping model can be useful for understanding distributed systems emerging from natural (e.g., biological and chemical) phenomena.

1.3 Designing Efficient Primitives for Fundamental Distributed Communication Problems

This thesis focuses on fundamental distributed communication primitives. We are interested in designing time-efficient, uniform and deterministic solutions to such problems.

Time Efficiency. The foremost concern of this thesis is the design of time-efficient solutions. Decreasing the running time of a solution generally also decreases its bandwidth and energy consumption. The natural time measured in the beeping model is in terms of rounds (see round complexity defined in Section 2.2).

Uniform Solutions. A second prime concern of this thesis is the design of *uniform* solutions, that is, not requiring any knowledge on the communication graph's parameters (even any upper bounds). Indeed, it is unrealistic to assume that such parameters are always available, especially when considering the ad-hoc nature of many wireless networks. These networks are not constrained by any a priori fixed structure. Instead, the communication graph highly varies with each network deployment.

Notice that, in the beeping model, algorithm design is easier if nodes know the communication graph's parameters. In particular, nodes may be designed to wait until one algorithm has terminated, before starting the subsequent one. The time to wait is a pre-computed time upper bound which generally depends on the graph parameters. On the other hand, without any information about these parameters, ensuring proper execution of consecutive algorithms is difficult: it is hard for a node to determine whether distant nodes have finished executing the current algorithm. As a result, uniform solutions are much harder to obtain than non-uniform ones.

Deterministic Guarantees. A third concern of this thesis is the design of *deterministic* solutions. Deterministic solutions are useful whenever random behavior is inappropriate or deterministic guarantees are required. For example, it may appear costly to incorporate random generators into weak devices. Moreover, the events in the system itself may not be necessary random, thus no randomness in the execution scheduling can be assumed. Furthermore, amongst existing works on the beeping model, the more difficult (for design) deterministic case has received less attention.

The problems considered in this work (defined in Section 2.3) all embody symmetry-breaking to some extent. It is well known that without any way to break symmetry (e.g., with unique identifiers, a conflict-resolving/asymmetric scheduler or an asymmetric communication graph), solving these problems deterministically is impossible [7]. We assume a synchronous (inherently symmetric) scheduler and an arbitrary communication graph (i.e., possibly also completely symmetric). Therefore, our deterministic solutions assume that nodes have *unique identifiers*, which is a natural and common assumption.

1.4 Contributions and Outline

The results of the thesis are presented in Chapters 4 through 7. In this section, we give an overview of these results. They all gravitate around the design of efficient distributed communication primitives. Obtaining efficient communication in the beeping model is challenging, as beeps have poor expressiveness (being unary

signals) and suffer from collisions. Overcoming these two difficulties requires interference control. Throughout this thesis, we leverage *symmetry-breaking* and *coding* techniques to achieve efficient interference control.

On the one hand, symmetry-breaking primitives (e.g., vertex coloring, maximal independent set, leader election or desynchronization) allow nodes to avoid (the interference produced by) collisions. By avoiding all collisions, nodes can straightforwardly communicate messages.
For instance, computing a 2-hop coloring (a basic and local symmetry-breaking problem) allows to assign different colors to nodes within distance 2. As a result, by communicating (perpetually) in order of colors, nodes avoid *sender-side collisions* (in which a node and at least one of its neighbors beep) and *receiver-side collisions* (in which a node listens and at least two of its neighbors beep). By avoiding these two types of collisions, one can implement *message-passing between neighboring nodes*.
Likewise, electing a leader (a global symmetry-breaking problem) allows to coordinate *network-wide messages* such as to avoid any interference.

On the other hand, coding techniques (e.g., group testing or superimposed coding) allow nodes to control the amount of interference (and its negative impact) during simultaneous communications.
For instance, nodes can use superimposed coding to transmit messages simultaneously. To do so, they beep and listen in a coded manner (determined by some codeword), which limits the negative impact of interference. More precisely, multiple codeword transmissions result in the OR-superposition (see Section 2.2) of these codewords. For a small number of codewords, this superposition is unique and can thus be decoded (despite collisions). By decoding it, nodes can extract all codewords (i.e., messages) sent in their neighborhood.

In essence, the methods provided in this thesis can be categorized as interference control on a *local scale* or a *global scale*. Additionally, some assume *global clocks* whereas others assume *uncoordinated local clocks* (see their description below).

Synchronous vs. Uncoordinated Starts. First, we give some preliminary information on the settings assumed to get the results. In the first part, we assume that nodes start at the same time (almost equivalently, nodes start at some arbitrary times or upon hearing a beep, whichever happens first). Then, in the second part, we assume a much harder setting, where nodes start at some arbitrary times and do not wake up upon hearing a beep. In both parts the communication graph is assumed to be general, and its parameters, such as the diameter D, maximum degree Δ and the network size n are unknown for most of our results. That is, the solutions are uniform.

1.4.1 Main Contributions

Our contributions are split between the two settings. The corresponding publications are [16, 52, 51, 15, 50].

Synchronous starts: For this setting, we give the first deterministic and uniform $(\Delta+1)$-vertex coloring algorithm (resulting in a deterministic and uniform maximal independent set algorithm), as well as the first deterministic $O(a)$-vertex coloring (where a is the arboricity of the communication graph) in the beeping model. In particular, most of the existing previous randomized algorithms either are non-uniform [38, 2, 96] or use more than $\Delta + 1$ colors [23]. These newly designed local symmetry breaking primitives are extremely efficient in sparse graphs. They allow interference control and are useful for simulating the message-passing model.

Later, improving on [60, 58, 39], we focus on global symmetry breaking and propose the first deterministic and uniform time-optimal leader election algorithm, as well as the first time-optimal randomized leader election solution (with anonymous nodes).

Finally, by leveraging group testing techniques, we present the first computationally- and time-efficient multi-broadcast algorithms, both deterministic and randomized, improving on [40, 42].

Uncoordinated starts: The only previous works in this setting are [38, 2]. We give the first primitive for simulating communication on the square graph, by using original coding theory tools. In particular, the primitive is deterministic. By leveraging it, we give the first randomized 2-hop desynchronization algorithm, improving on the pioneering desynchronization algorithm of [38]. Furthermore, using this new desynchronization algorithm, the message passing model can be simulated even in this setting.

1.4.2 Detailed Outline

Some preliminaries are addressed in Chapters 2 and 3. Chapter 2 defines the beeping model, important notations as well as the problems considered in this thesis. Chapter 3 gives a general overview of previous works in the beeping model.

Beeping Model with Synchronous Starts. In Chapters 4 to 6, the focus is on deterministic and uniform solutions. We give algorithms for problems with either no known solutions or improve upon the existing ones, while providing their time optimal versions. These results also contribute to a better understanding of how the beeping model relates to both $\mathcal{CONGEST}$[1] (i.e., message-passing) with 1-bit messages and the classical wireless radio network model with collision detection[2].

- Our first result (Chapter 4) considers *local symmetry-breaking problems* - more precisely *vertex coloring* and *maximal independent set*. In the $(\Delta + 1)$-vertex

[1]The $\mathcal{CONGEST}$ model [90] assumes that nodes communicate via a graph-based message-passing infrastructure. Different messages of bounded size can be sent to different neighbors, and nodes receive the full content of all the incoming messages and their incoming port (i.e., the link identifier through which the message was received).

[2]Nodes communicate $O(\log n)$ size messages via local broadcast, according to some undirected communication graph. Collisions produce destructive interference (thus all message content is lost upon collisions). If collision detection is available, then algorithms designed in the beeping model can be straightforwardly translated to this model.

coloring problem, each node computes a color in $\{1, \ldots, \Delta + 1\}$ such that no two neighbors have the same color. In the maximal independent set (MIS) problem, nodes compute a set of nodes which is both independent (no two neighbors are in the set) and maximal (no node can be added to the set while satisfying the independence property). These two problems are major problems in distributed computing. Coloring is a central problem in wireless and radio networks in general. The computed colors can be used as local identifiers, allowing in turn to solve many other problems. On the other hand, an MIS can be used to decompose the network into clusters of radius 1, such that each node in the MIS is a cluster head. Clustering is a general approach for task allocation and resource sharing.

We give deterministic algorithms for these two problems, as well as for their 2-hop variants: 2-hop coloring and 2-hop MIS. Then, we present an algorithm which uses a 2-hop coloring solution to simulate a classical message-passing (i.e., $\mathcal{CONGEST}$) model with B-bit messages in the beeping model, for an $O(\Delta^4 B)$ multiplicative overhead. Finally, we present the first deterministic algorithm for $O(a)$-coloring (where a is the arboricity of the communication graph). All solutions presented in this chapter are extremely efficient in sparse graphs.

- Our second result (Chapter 5) focuses on *leader election* (LE). Conversely, LE is a global symmetry-breaking problem. In LE, a single node should consider itself the leader, and all other nodes should be aware of the leader's identifier. Leader election is a longstanding problem in distributed computing. Electing a leader allows to decide on a node with authority over the network. That node can then coordinate all other nodes in order to control network-wide interference, or act as the root for a spanning tree in order to gather information efficiently from all over the network.

  We present an optimal $O(D + \log n)$ leader election algorithm in the beeping model, which is also an optimal LE algorithm for the stronger (and more traditional) radio network model with $O(\log n)$ bit messages and collision detection. Additionally, this result shows, somewhat surprisingly, that leader election has the same asymptotic complexity in the beeping model and in $\mathcal{CONGEST}$ with 1-bit messages, allowing to conclude that beeps may provide time-efficient communication. Finally, since LE is an important primitive, the improved leader election algorithm is an essential component in improving information dissemination solutions (see [42] as well as Chapter 6).

- Our third result (Chapter 6) considers information dissemination in the beeping model - more precisely the *multi-broadcast* problem. In this problem, an unknown number of source nodes need to disseminate their messages to all nodes of the network, in such a way that all nodes are aware of all messages and the corresponding source's identifiers. As a primitive, multi-broadcast is a common abstraction of communication over the network. By leveraging this primitive, multiple nodes can broadcast simultaneously over the network in a time-efficient manner. Moreover, it allows to implement node-to-node data

transfer.

The previous known deterministic multi-broadcast algorithm [42] has optimal time complexity, but needs excessive computational power for the required pre-processing computation. We start by devising an optimal deterministic multi-broadcast algorithm with improved pre-processing computation, when there are few sources. However, this algorithm still uses a high amount of computational power, exponential in the number of sources. Then, we propose nearly-optimal deterministic and randomized multi-broadcast algorithms. These are optimal for most ranges of the parameters. This time, both algorithms use only a reasonable (polynomial) amount of computational power for pre-processing.

Beeping Model with Arbitrary Starts. In the second part of the thesis, we assume that nodes wake up in an uncoordinated manner. This setting is significantly harder, but in turn the results are more widely applicable. Moreover, they may inspire design techniques for future dynamic [22] or self-stabilizing [45] algorithms in the beeping model. Indeed, consider a dynamic network in which nodes join and leave repetitively (or equivalently are victims of transient faults, in the context of self-stabilization). Then, solutions that consider synchronous starts are absolutely helpless when faced with such dynamics, whereas dealing with uncoordinated starts means dealing in part with the difficulties of the dynamic setting. Additionally, notice that biological distributed systems, relying on energy efficient but extremely basic communication mechanisms, are surprisingly tolerant to unexpected situations which might arise due to faults or due to the dynamic nature of the system. Hence, studying the impact of limited communication in the arbitrary start setting may also help understanding biological systems.

Two main results are obtained in this setting. They are summarized below and are presented in detail in Chapter 7. Both results consider the square of the communication graph (obtained by adding to the given graph additional edges between distance 2 nodes). This is a clear departure from previous works in this setting, which studied problems needing no information beyond the 1-hop neighborhood of a node. Whereas here, we consider problems in which nodes need to gather information from their 2-hop neighborhood. The difficulty lies in that the nodes must somehow coordinate with their neighbors to propagate a beep to distance 2 despite the basic communication mechanism and uncoordinated starts.

- Our fourth result is a deterministic (but non-uniform) primitive allowing nodes to communicate to their 2-hop neighbors. Such a primitive is particularly relevant in wireless radio networks. Indeed, interference can remain an issue even if a method allowing to break symmetry within the immediate neighborhood is available. For instance, a node can still suffer from collisions if two of its neighbors (at distance 2 from each other) communicate simultaneously: this is called the *hidden terminal* problem. By using this primitive to translate such symmetry-breaking methods to 2-hop neighborhoods, these collisions can be taken care of.

Although this primitive is easily available in the synchronous starts setting, it was unknown whether nodes could properly communicate to their 2-hop neighborhood if nodes start in an uncoordinated manner. The difficulty lies in the lack of a coordinated view of time between the nodes, which makes the simple interference mechanism of beeps extremely hard to deal with. Due to this difficulty, nodes are very limited and seem unable to convey complex information.
By developing a variant of *superimposed codes*, we show how this difficulty can be overcome to obtain the 2-hop communication primitive, albeit with a slight drawback: nodes at distance 2 receive a beep after a certain time delay.

- Our fifth result is a randomized and non-uniform *2-hop desynchronization* algorithm in the uncoordinated starts setting. In the 2-hop desynchronization problem, nodes seek to determine infinite disjoint (in its 2-hop neighborhood) sequences of (global) rounds, for the sake of avoiding collisions. Importantly, a 2-hop desynchronization solution is an efficient medium access control method. Such methods are essential components in wireless networks, since they can be used to establish a reliable MAC layer. This layer controls interference on a local scale, which allows nodes to implement message-passing.

  To obtain a 2-hop desynchronization solution, we apply the 2-hop primitive to an existing 1-hop desynchronization algorithm [38], by controlling the number of simultaneous communications and dealing with the delay in communication introduced by the 2-hop communication primitive. Using the 2-hop desynchronization solution to avoid collisions, we implement reliable message-passing primitives in this harsh beeping model.

Chapter 2

Preliminaries

2.1 Model

In the beeping model, time is divided into synchronized rounds (with the exception of [69]). The *communication graph* is denoted by a simple static connected undirected graph $G = (V, E)$, where V is the node set (i.e., processes) and E the edge set, representing possible communication between processes. The *network size* $|V|$ is denoted by n, the *diameter* by D and the *maximum degree* by Δ. Time is divided into discrete time intervals, called (*global*) *rounds*. First, we describe the most general beeping model which we denote by $\mathcal{BEEP}_U$. A node wakes up spontaneously at an *arbitrary* round (arbitrary time offset). From this starting round onwards, the node is said to be *awake*, and then in each subsequent round, synchronously with other awake nodes, it executes the following steps. First, the node beeps (instruction $BEEP$ in algorithms) or listens ($LISTEN$ in algorithms). Beeps are transmitted to all (awake) neighbors of the beeping node during the round. Then, if the node listens (in the previous step of the same round), it knows whether or not at least one of its neighbors beeped (during the previous step of the same round). Finally, the node performs local computations.

Local and Global Rounds. For any given global round r, an awake node v in r knows only the *local* round value r_v (round value relative to node v's wake-up time). For any local round r_v, v is unaware of the global round. Thus, any two nodes u and v may have uncoordinated local clocks (see Figure 2.1). For the sake of analysis, for any given node v, a function g_v is defined such that for any local round $r_v \geq 1$, $g_v(r_v)$ denotes the global round corresponding to r_v. Additionally, g_v has an inverse function, denoted by g_v^{-1}.

Global rounds: from the first wake-up	38	39	40	41	42	43	44	45	46	47	48	49	50	51	52	53	54
Node v's local rounds									1	2	3	4	5	6	7	8	9
Node u's local rounds			1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

Figure 2.1 – Local rounds can be arbitrarily different in $\mathcal{BEEP}_U$, due to adversarial wake-up.

Identifiers. Nodes have unique identifiers (IDs). This property is essential in order to break symmetry in deterministic algorithms. The *identifier* of a node

$v \in V$, id_v, is an integer from $\{1, \ldots, L\}$ where L is some upper bound (dependent on G) on the identifiers unknown to nodes. Then, the *maximum length* over all identifiers in G is $\lceil \log L \rceil$ (also unknown). For simplicity, we make the common assumption (in most chapters) that identifiers have logarithmic (in n) length, i.e., the ID space is $\{1, \ldots, N\}$ where $N = n^c$ for some unknown constant $c > 1$.

Wake-up Assumptions. The wake-up assumption can differ. More precisely, the *uncoordinated wake-up* assumption, in which nodes wake up arbitrarily (denoted by $\mathcal{BEEP}_U$), is the most general but two other simpler, more restrictive variants are more typically considered. In *wake-on-beep* ($\mathcal{BEEP}_W$) nodes wake up upon hearing a beep or arbitrarily, whichever comes first. In *synchronous wake-up* ($\mathcal{BEEP}_S$) nodes all wake up at the same time.

Depending on the wake-up assumption, nodes' local clocks either are synchronous (i.e., a global clock), locally almost synchronous (i.e., differ by less than a constant between neighbors) or uncoordinated (i.e., arbitrarily different). The first two settings are presumably stronger than the third one. Indeed, synchronous local clocks can be used to convey more information than just a beep: for example, an algorithm can distinguish beeps in odd and beeps in even rounds, which is a simple trick used to simulate communication on the square communication graph in $\mathcal{BEEP}_S$ (cf. Section 4.5). Since locally almost synchronous clocks can be used to simulate synchronous local clocks [1], they similarly allow a beep to convey additional information. However, uncoordinated local clocks cannot a priori be used to convey information beyond a simple beep since neighbors' clock values are arbitrarily different. It is unclear how the first two wake-up assumptions can be (or if they can even be) simulated using uncoordinated local clocks.

2.2 Definitions and Notations

In this section, we give definitions and notations that will be used throughout this thesis.

Graphs. The *square of the communication graph* is denoted by $G^2 = (V_2, E_2)$, where $V_2 = V$ and $E_2 = E \cup \{\{v_1, v_2\} \in V^2 | \exists u \in V \setminus \{v_1, v_2\}$, s.t. $\{v_1, u\}, \{u, v_2\} \in E\}$. For any given node v, its one-hop neighborhood in G is denoted by $\mathcal{N}(v) = \{v\} \cup \{u \in V | \{v, u\} \in E\}$ and its 2-hop neighborhood (i.e., its 1-hop neighborhood in G^2) by $\mathcal{N}_2(v) = \{v\} \cup \{u \in V | \{v, u\} \in E_2\}$. Node v is included in both sets. For a node $v \in V$, the *neighbors* of v are $\mathcal{N}^*(v) = \mathcal{N}(v) \setminus \{v\}$ and the 2-hop *neighbors* of v are $\mathcal{N}_2^*(v) = \mathcal{N}_2(v) \setminus \{v\}$. Subsequently, the degree of a node v in G is $d(v) = |\mathcal{N}^*(v)|$ and its 2-hop degree (i.e., its degree in G^2) is $d_2(v) = |\mathcal{N}_2^*(v)|$.

The *distance* between two nodes u and v in G is $dist(u, v)$. Equivalently, the square graph of G is the graph $G^2 = (V_2, E_2)$, where $V_2 = V$ and $E_2 = \{\{u, v\} | u, v \in V, dist(u, v) \leq 2\}$. $G[R]$ 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. Arboricity can equivalently [84] be defined as a measure of density: $a = \max_{R \subseteq V, |R| \geq 2} \frac{|E_G[R]|}{|R|-1}$.

Formal Language. We use the terminology of formal language theory and focus on the alphabet $\{0, 1\}$. The empty word is denoted by ϵ. The operator $\|$ is for the *word concatenation*. For any positive integer i, 0^i (respectively, 1^i) denotes the concatenation of i symbols 0's (resp., 1's) (where $0^0 = \epsilon$). The *length* of a word x is denoted by $|x|$, $x[j]$ denotes the j^{th} bit of x and $x[i, j]$ the *factor* of x, from the i^{th} to the j^{th} bit. Let x and y be two words (of possibly different lengths), x is a *prefix* (resp., *proper prefix*) of y if there exists a word (resp., non empty word) z such that $x \| z = y$. Moreover, x is *greater* (in lexicographical order) than y, denoted by $x \succ y$, if y is a proper prefix of x, or if $x[j] > y[j]$ for the first differing bit j (even if $|x| < |y|$).

The following operations are illustrated using Figure 2.2. For any two words x and y of the same length, we define the (bitwise OR) *superposition* of x and y (and say that x and y are (OR) *superposed*) as the binary word w of length $|w| = |x|$ such that $\forall i \in \{1, \ldots, |w|\}$, $w[i] = 0 \Leftrightarrow x[i] = y[i] = 0$. We naturally extend the superposition to the case of several words of the same length.

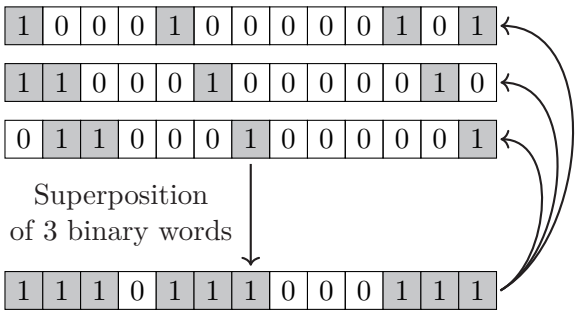


Figure 2.2 – Superposition and decomposition on binary words.

Additionally, for any two words x and y of the same length, x is said to be *included* in y if $\forall i \in \{1, \ldots, |x|\}$, $x[i] = 1 \Rightarrow y[i] = 1$. Finally, for any binary word w and a set of words W, we define a *decomposition* of w in W, as a subset S of W for which the superposition of all words is w. The decomposition of a word is not guaranteed to be a function.

System Definitions. We adopt the usual definitions for the system/algorithm. The *state* of a node is defined by the current values of its variables. A *configuration* is a vector of the states of all the nodes. The *algorithm* is defined as a transition function τ on states. In this work, we consider the beeping model, therefore we also consider the *synchronous scheduler*: in each round, nodes apply τ on their state. As a result, the algorithm can be straightforwardly translated to a transition function τ' on configurations. An *execution* is a sequence of configurations where consecutive configurations are obtained by applying τ'. A *terminal configuration* is a configuration that stays unchanged when applying τ' (it is repeated in an execution).

Termination is obtained when a terminal configuration has been reached. A variable *var* of a node v is explicitly associated to v using a subscript var_v.

A *problem* is given as a first order predicate over executions. An algorithm is said to *solve a problem* if each execution terminates and satisfies the predicate of the problem specification. In this thesis, such predicates can be naturally obtained from the problems' definitions given in Section 2.3. The *round complexity* (time complexity) of an algorithm is the number of rounds needed to reach the first terminal configuration in the worst case. An algorithm is said to be *locally termination detecting*, or simply *locally terminating*, if for any given node v, v is aware if it has reached a terminal state. An algorithm is said to be *uniform*[1] in a parameter p if the algorithm solves the considered problem for all values of p with the same transition function. In other words, the algorithm is not given p and is unable to infer it from the information it receives. For example, in a uniform (in n) algorithm, nodes do not know the size n of the network, neither can they deduce it from their identifier.

Defining α-encoding. Introduced in [25], an *α-encoding* is a tool which allows to compare integers (identifiers) bit by bit in a uniform manner (i.e., to compare their binary values). Indeed, when using α-encodings (of integers from $\{1, \ldots, N\}$), such algorithms do not need to know the binary values' lengths (depending on $\log N$) to compare them bit by bit (see Lemma 1). Importantly, *uniform comparison of IDs* is an essential component in order to perform deterministic symmetry-breaking. As a result, α-encodings of IDs (α-IDs) are used in the local symmetry-breaking and leader election solutions of Chapters 4 and 5.

Definition 1. *Let i be a positive integer and bin its binary representation. The α-encoding of i, denoted by $\alpha(i)$, is $1^{|bin|} \parallel 0 \parallel bin$.*

The α-encoding of integer i is made up of two parts, as explained in Figure 2.3. 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.

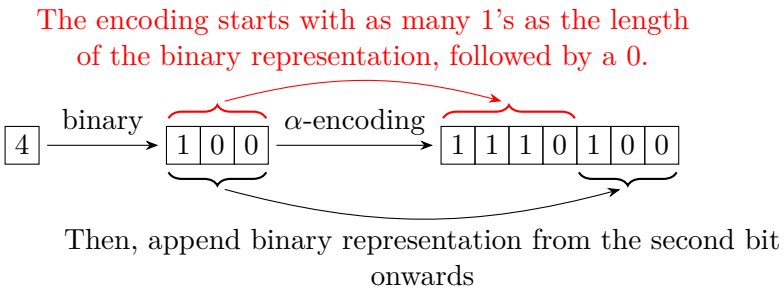


Figure 2.3 – Description of α-encoding

[1] It is known that termination detection is easy in a synchronous setting whenever particular parameters related to the size of the communication graph are known, i.e., non-uniform terminating algorithms are easier to construct than the uniform ones.

α-encoding preserves the order between two integers.

Lemma 1 ([25]). *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.*

Modulo Operation. 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 $a \mod k = b \mod k$.

2.3 Problem Definitions

In this section, we give precise definitions of the problems studied in this thesis. These problems are fundamental building blocks in distributed computing, and are also often used to understand the relation between the different distributed computing models.

Local Symmetry-Breaking Problems (MIS and Vertex Coloring). The following two problems are considered to be local symmetry-breaking problems, as two neighboring nodes cannot have the same output. Thus, obtaining a solution requires breaking symmetry between neighboring nodes.

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 (MIS defines the MIS problem specification). A *c-coloring col* is a function from V into a set of colors $\{1, \ldots, c\}$ such that $\forall (u, v) \in E$ $col(u) \neq col(v)$ (defining the c-coloring problem specification). These two problems are tightly connected. Notice that in a vertex coloring solution, nodes with the same color constitute an independent set. By starting with an empty set and adding non-conflicting nodes in order of increasing colors, an MIS is obtained. Inversely, a set of disjoint MIS that cover the network can be used to obtain a coloring, since the same color can be given to all nodes of an MIS (and there are at most $\Delta + 1$ MIS). In addition, we define the 2-hop variants of these problems: a 2-hop MIS (respectively, 2-hop coloring) of G is an MIS (resp., coloring) of its square graph G^2.

Leader Election Problem. In the *leader election* (LE) problem, each node has a boolean variable, indicating a *leader* or a *non-leader* state. During an execution, there is never more than one leader (*safety* property). Initially, all nodes are non-leaders. Every execution terminates, and at the termination there is exactly one leader.
Now we give auxiliary definitions. First, we define *eventual leader election*, where the algorithm terminates but no node can detect this. Then, we define *terminating leader election*, where the algorithm terminates and all nodes detect when there remains a single candidate node (the leader). Finally, we define *explicit* leader election (when nodes have unique identifiers): a terminating leader election in which all nodes know the elected leader's identifier at the termination.

Information Dissemination Problems (Multi-Broadcast). Let S be a subset of k nodes (for some $k > 1$) called *sources* and having (possibly identical) messages in $\{1, \ldots, M\}$, where M is unknown to all nodes. For any node v, m_v denotes its single message. If v is not a source then $m_v = \epsilon$. Equivalently, m_v refers to the binary representation of length at most $\lceil \log M \rceil$. In the *multi-broadcast* (with provenance) problem, all nodes must receive the message of each source with its ID. In other words, they must compute the set $\{(m_v, id_v) \mid v \text{ is a source } \}$. The *gossiping* problem is a variant of the multi-broadcast problem in which all nodes are sources.

2-hop Desynchronization Problem. In the *2-hop desynchronization* problem, every node has to determine a sequence of (global) rounds, disjoint from those of its 2-hop neighbors. The sequence should form an *arithmetic sequence* from some point on. The difference between consecutive values of that arithmetic sequence (i.e., the *common difference*) is denoted by T, and is said to be the *period* of the sequence. In a sense, if a node beeps according to such an arithmetic sequence, then T is the period between that node's consecutive beeps. If two nodes determine disjoint sequences from a certain round onwards, they are said to be *desynchronized* with regard to T (see Figure 2.4). Notice that, as illustrated in Figure 2.4, nodes can only decide on such sequences (of global rounds) by using their local view of time (local rounds).

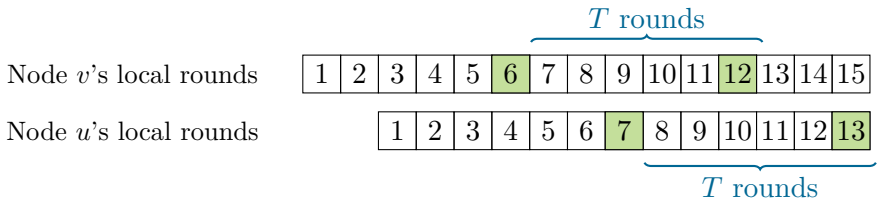


Figure 2.4 – Disjoint sequences of rounds (the colored rounds) with period $T = 6$ of nodes u and v.

Chapter 3

Related Work

The results concerning the beeping model fall into different categories. First, we consider the related work for the beeping model with a single-hop communication graph. This setting is equivalent to that in which nodes communicate via a noiseless multiple access channel. Traditional results on multiple access channels assume ternary or binary feedback: in each round, nodes know whether no node communicated, a single one did or whether a collision happened (respectively, know whether a single node communicated or detect communication noise). Second, we consider the related work for the more general setting, that is, with a multi-hop topology, starting with local scale problems, followed by global scale problems.

3.1 Beeping in a Single-hop Network.

Authors of [70] were the first to consider the single-hop beeping model, which they motivated by drawing connections with circuit-based algorithm implementations and more generally shared communication channels. Nodes are assumed to wake up at the same time, and have identifiers. In [70], it is shown that there is an equivalence between the *membership problem* - in which nodes need to find out the IDs of a subset of active nodes - and *conflict resolution problem* - in which every active node needs to use the channel alone at least once - in the beeping model. Moreover, [70] presents a deterministic algorithm solving these problems, which scales if multiple channels are available, as well as a randomized lower bound.
[68] considers a slightly different setting, the fault-prone beeping multiple access channel, in which there is a non-zero probability in each round that a fault occurs in the channel. In a faulty round, nodes hear nothing regardless of their behaviors. A deterministic algorithm provides a global clock with arbitrarily small constant error probability - unavoidable due to the channel faults - which is then used to obtain a logarithmic time *consensus* algorithm.
[30] consider the *renaming problem*, in which nodes hold no IDs and should be given unique numbers in $\{1, \ldots, n\}$ where n is the network size. An $O(n \log n)$ expected time Las Vegas[1] (LV) renaming algorithm is presented first, requiring that n be known, followed by a Monte Carlo[2] (MC) renaming algorithm which does not require the knowledge of n.

Counting the number of nodes is considered in [19, 26]. The solution in [19] provides an $(1+\epsilon)$-approximation of the network size using $O(\log \log n + \log \frac{f}{\epsilon^2})$

[1] A Las Vegas (LV) algorithm is a randomized algorithm that guarantees a correct output and has some probability at least $p(T)$ of finishing within a finite number of rounds T.

[2] A Monte Carlo (MC) algorithm is a randomized algorithm with a deterministic time guarantee and that has a correct output with some probability at least p.

rounds with probability $1 - \frac{1}{f}$ (for arbitrary ϵ and f), and is almost tight. In contrast, the solution in [26] provides the exact value of the network size n with high probability[3] (w.h.p.) using $O(n)$ rounds if collision detection is assumed, and $O(n \log n)$ otherwise.

In a slightly different mindset, [61, 62] present leader election solutions for a clique of beeping finite state machines. Instead of focusing solely on time efficiency, [61] first gives a randomized LV state-optimal leader election algorithm. Then, in a second time, [61] provides a faster randomized LV leader election algorithm, but in return the algorithm uses a less than optimal number of states. Additionally, a clique of beeping finite state machines is shown to be able to simulate a logarithmic space turing machine with a given error bound. [62] follows up on [61] by considering leader election in single-hop networks and MIS in multi-hop networks with a computational noise modelization - a node's state machine might transition erroneously due to the computational noise.
Finally, [17] present a low layer communication protocol such that nodes can transmit encrypted messages to a designated sink node.

3.2 Beeping in a Multi-hop Network.

Local Symmetry-breaking. The beeping model was first considered with the uncoordinated wake-up assumption ($\mathcal{BEEP}_U$), for symmetry-breaking problems. The pioneering paper of [38] proposes a randomized $O(\Delta \log n)$ solution to *interval coloring* - or equivalently to the *1-hop desynchronization* problem. Nodes solve desynchronization by generating pulses periodically with a fixed common period T such that pulses of neighboring nodes are evenly distributed throughout the time period T. Clearly, the intervals centered on the pulses of nodes constitute an interval coloring. Such solutions are prime examples of symmetry-breaking, since they allow a node to choose a different interval than its neighbors. Following which [1, 3, 2] propose an efficient $O(\log^3 n)$ randomized LV algorithm for the MIS problem given an upper bound N on the network size. Importantly, both results in $\mathcal{BEEP}_U$ require a priori knowledge: in particular, [1, 3, 2] prove that MIS takes $\Omega(\sqrt{\frac{n}{\log n}})$ rounds if no upper bound on the network size is known. Both results are also probabilistically self-stabilizing (i.e., converges to a correct configuration from any given initial configuration, with some probability).

All the following results consider $\mathcal{BEEP}_S$ or $\mathcal{BEEP}_W$ only. [1, 3, 2] give an efficient $O(\log^2 n)$ randomized LV algorithm for the MIS problem assuming collision detection capabilities. [96] follow up on this result with an $O(\log n)$ w.h.p. time optimal randomized LV MIS algorithm if collision detection is available, which can be transformed into an $O(\log^2 n)$ w.h.p. time optimal randomized MC MIS algorithm without collision detection.
[27] presents randomized coloring, 2-hop coloring, MIS and 2-hop MIS solutions, and in particular improves on the constant for the MIS algorithm of [96], from an extremely large constant to a more practically relevant constant.
Finally, [67] gives a randomized MIS algorithm with good "local" complexity - with

[3] With high probability (w.h.p.): with error probability upper bounded by $n^{-\theta(1)}$.

dependence only on a node's degree and the intended error bound - instead of the more common "global" complexity which is defined by the slowest node and thus typically depends on the network size n.

Some results consider topologies commonly assumed in wireless network literature. [100] focuses on *backbone construction* and gives approximation algorithms for minimum dominating set and minimum connected dominating set in unit disk graphs. [65] gives a randomized one-round algorithm resulting in a polylogarithmic approximation of the maximum independent set in graphs with polynomially bounded-independence.

Leader Election, Multi-Broadcast and Data Aggregation. The first global scale problem studied in the beeping model was *leader election*. Although it is necessary in order to solve global problems, it is not clear at first that beeps can be used to communicate accross the network because of the interference. [60] introduced *beeping waves*, a technique for communicating a message (using beeps) from a designated root. They also proposed a leader election algorithm which selects a reduced number of candidates that then detect the presence of other candidates using these beeping waves and randomized sequences to detect collisions, resulting in a randomized MC solution with $O((D + \log n) \cdot (\log^2 \log n))$ w.h.p. round complexity. Since leader election in the beeping model requires $\Omega(D + \log n)$ rounds, the above complexity is nearly optimal.

The first deterministic $O(D \cdot \log n)$ round solution is presented in [58]. This solution is uniform, i.e., does not require any parameter knowledge. An important building block in their solution is the Balanced Execution Technique, which allows to sequentially execute uniform algorithms without any parameter knowledge in a locally synchronous manner.

By combining the beep wave technique and the deterministic leader election solution above, [40] presents algorithms for *information dissemination* and *data aggregation*. More precisely, they give algorithms for the broadcast, multi-broadcast (with and without provenance) and the gossiping problems. In part due to the non-optimality of the leader election in [58], the multi-broadcast and gossiping solutions are non (but nearly) optimal.

Drawn by the similarities in communication between the beeping model and swarms of fireflies, [6, 64] study *synchronization* in the beeping model. [6] considers the wake-on-beep model and shows that in certain conditions, using the "averaging rule" allows the network to converge to a single periodic beep. This solution relies heavily on the wake-on-beep assumption, whereas it is also interesting to study synchronization in $\mathcal{BEEP}_U$. Then, [64] consider self-stabilizing synchronization with byzantine nodes, but augment the beeping model with the capability to count the number of neighboring beeping nodes. As a result, the work gives little intuition regarding self-stabilizing byzantine synchronization in the classical beeping model.

Finally, the problem of *counting* the number of nodes over the network was first considered in dynamic graphs in [89]. Algorithms presented in a stronger model are adapted to the beeping model and result in an $O(n^2 \log n)$ round deterministic exact counting algorithm.

With a different perspective, [79] consider the problem of computing local sums and

gives a randomized $(1+\epsilon)$ approximation algorithm (computing these local sums). Interestingly, this algorithm can be used to construct a randomized approximation algorithm for random walk distribution which can then be adapted to approximation algorithms for PageRank and global sum. The latter randomized approximation algorithm for global sum can also be used to approximate (within an $(1+\epsilon)$ factor) the number of nodes in the network.

Some Remarks. Some unusual variants assume a very different setting. [54] considers mobile robots moving on a graph and communicating using beeps and [69] considers an asynchronous beeping model with bivalent beeps: nodes can use loud beeps or soft beeps. In the latter, bivalent beeps are required to deal with the impossibility results caused by the asynchrony assumption. Multiple soft beeps are indistinguishable from a loud beep.

Chapter 4

Local Symmetry-Breaking Methods

Chapters 4 through 6 consider the beeping model in which nodes wake up simultaneously ($\mathcal{BEEP}_S$). Solutions obtained assuming synchronous starts can be easily translated to $\mathcal{BEEP}_W$ (where nodes wake up at some arbitrary times or upon hearing a beep, whichever comes first) with an additive $O(D)$ time overhead. This overhead is caused by the nature of the wake-up assumption. The adversary can wake up a single node, which will then wake up the whole network starting with its neighbors, such that the last node inevitably wakes up $O(D)$ rounds after the first one.

In both models, algorithms can use the synchronous nature of the rounds (i.e., the high synchronization between neighboring nodes) to convey collision-tolerant information through beeps. However, doing so has a quantifiable, negative impact on the time complexity. This work studies the efficiency of using beeps in such a way.

In addition, the non-destructive interference caused by beep collisions (i.e., two or more beeps cannot be distinguished from a single beep but can be distinguished from zero beeps) remains a difficulty. To deal with this difficulty, we study local symmetry-breaking problems, such as vertex coloring and maximal independent set. Solving the 2-hop variants of these problems provides the (local) interference control needed to implement traditional message communication primitives (i.e., message-passing between neighboring nodes), as is shown in Section 4.6.

4.1 Introduction

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. More specifically, a coloring can be used to allocate resources that cannot be shared by neighboring nodes. For example, a coloring can be used to allocate access to the communication medium and avoid interference. Likewise, nodes in an MIS can act as cluster heads in order to coordinate actions, and participate in a network backbone construction. In particular, interference can be controlled by limiting communication to these cluster heads.

Therefore, the MIS and coloring problems serve as important primitives for algorithm design in the beeping model and have naturally received a lot of attention (see Section 4.1.1). 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 (see Section 4.1.1). In this work, we are interested in designing deterministic algorithms having efficient time complexity.

Sparse Communication Graphs. Because of the ad-hoc nature of wireless network, communication graphs do not follow a fixed structure. Graphs with low edge density are said to be *sparse*. The *maximum degree* and the *arboricity* of a graph (see definitions in Section 2.2) 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 [56]. In particular, graphs embedded in some surface, for example the plane, have low arboricity. In this section, we leverage the sparsity of the communication graph in order to design efficient symmetry-breaking algorithms.

4.1.1 Specific Related Work

In [27], 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 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 a collision. In [38], the authors present the first (non-uniform) coloring algorithm for the beeping model (more precisely, for $\mathcal{BEEP}_U$). It outputs a correct coloring after $O(\Delta+\log n)$ rounds w.h.p. Following this paper, randomized MIS and coloring algorithms were designed for $\mathcal{BEEP}_S$ with collision detection, in a series of papers [2, 96, 23]. These algorithms achieve optimal round complexity, but assume collision detection. Moreover, the resulting colorings often employ more than $\Delta+1$ colors. These algorithms can be translated to the weaker $\mathcal{BEEP}_S$ with no collision detection with an $\Omega(\log n)$ multiplicative factor.

Schneider and Wattenhofer [94] solve *deterministic* MIS in the radio network model with collision detection. Although the term "beeping model" does not appear in [94], the presented algorithm straightforwardly works in $\mathcal{BEEP}_S$. 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 Section 2.3) 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 algorithms are compared below (see respectively Figures 4.1 and 4.2). The only deterministic algorithms are those in [94] and in the present paper. Some algorithms require K, an upper bound on Δ.

Table 4.1 – MIS algorithms in $\mathcal{BEEP}_S$

Reference	Time complexity	Comments
[96]	$O(\log^2 n)$ w.h.p.	Anonymous nodes
[94]	$O(\log n)$, deterministic	Growth bounded graphs
Our work	$O(\Delta^2 \log n + \Delta^3)$, deterministic	Scales for $\Delta = O(\log^c n)$
Our work	$O(a^2 \log^2 n + a^3 \log n)$, deterministic	Scales for $a = O(\log^c n)$

Table 4.2 – Vertex coloring algorithms in $\mathcal{BEEP}_S$

Reference	Time complexity	Number of colors used
[23]	$O(\Delta \log n + \log^2 n)$ (w.h.p.)	$\Delta + \log n$ colors
[23]	$O(K \log^2 n)$ (w.h.p.)	$K + 1$ colors
[38]	$O(\Delta + \log n)$ (w.h.p.)	$O(K)$ colors
Our work	$O(\Delta^2 \log n + \Delta^3)$	$\Delta + 1$ colors
Our work	$O(a^2 \log^2 n + a^3 \log n)$	$\Delta + 1$ colors
Our work	$O((a^2\Delta^2 + a^\mu \Delta^4) \cdot \log^2 n + a^3\Delta^3 \log n)$	$O(a)$ colors

4.1.2 Algorithm-related Definitions

In the beeping model, algorithms 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 algorithms, such loops are nested. In this section, 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 algorithms (see Section 2.2) 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 [58], with the "Balanced Execution Technique" (BET). However, BET only guarantees $\mathcal{L}_1$-synchronization points. In Section 4.7, 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 algorithms in $\mathcal{BEEP}_S$.

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

4.1.3 Auxiliary Problems

The problems defined below are used to obtain a vertex coloring algorithm. The ruling set allows for limited symmetry-breaking capabilities, and a coloring can be obtained by gradually reducing the defect of a defective coloring.

Ruling Set. A set $J \subseteq V$ of vertices is said to be a (t, s)-*ruling set* [9], if for any two vertices $u, v \in J$, $dist(u, v) \geq t$, and for any vertex $v \in V \setminus J$, 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.

Defective Coloring. Any given function $colorD$ is a *d-defective c-coloring* [14] if $\forall v \in V$, $colorD(v) \in \{1, \ldots, c\}$ 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.

4.2 Ruling Set Algorithm and Competition Graphs

Ruling sets serve as building blocks to construct complex algorithms. They have been used to compute MIS [9] and colorings [95, 93]. 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 subsequent Section 4.3, how ruling sets can still be used to design an efficient coloring algorithm.

In this section, we introduce a competition algorithm (*RulingSet* - Algorithm 1 in Section 4.2.1) computing a $(2, O(\log N))$-ruling set. This algorithm can be considered as a variant of the ruling set algorithm from [9]. 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 algorithm with a similar behavior. To prove correctness (Section 4.2.2), we use competition graphs, which are directed graphs that serve to model the behavior of competition algorithms and help analyzing them. They were first used in [66], but in association with a non-deterministic elimination process. As we are interested in uniform deterministic algorithms, we use the nodes' α-IDs (see definition in Section 2.2) to label the edges of a competition graph with integer values, and these values determine a deterministic elimination process. The resulting labeled competition graphs allow to compute the surviving nodes in a convenient way.

4.2.1 Uniform Competition Algorithm 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-identifier $\alpha(id_v)$ (or $\alpha(v)$ for short, by notation abuse) and outputs a boolean value $survived_v$. We prove that the output is a $(2, O(\log N))$-ruling set (Theorem 6).

Algorithm 1 *RulingSet*

```
1: IN: id: Integer  OUT: survived: Boolean value
2: survived := true, α := α(id)                          ▷ Get α-ID
3: for round r := 1 ; r ≤ |α| ; r++ do     ▷ r is incremented after each iteration
4:     if α[r] = 1 then
5:         BEEP                              ▷ Consider the r^th most significant bit
6:     else
7:         LISTEN
8:         if beep heard then                ▷ If a neighbor has a greater identifier
9:             survived := false
10:            EndAlgorithm
11: EndAlgorithm                                          ▷ No beep heard
```

The following lemma is straightforward.

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

4.2.2 Correctness Analysis of Algorithm 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* algorithm, but it applies to any competition algorithm. G_{dag} is defined as $(V, E_{dag}, label)$, where E_{dag} is the set of directed edges and $label$ an edge labeling function. G_{dag} is constructed from the α-IDs, encoded on a maximum of $2l_{max} + 1$ 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 $i \in \{1, \ldots, 2l_{max} + 1\}$ such that $\alpha(u)[i] = 1$ and $\alpha(v)[i] = 0$, 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. It is straightforward to prove that G_{dag} is a directed acyclic graph.

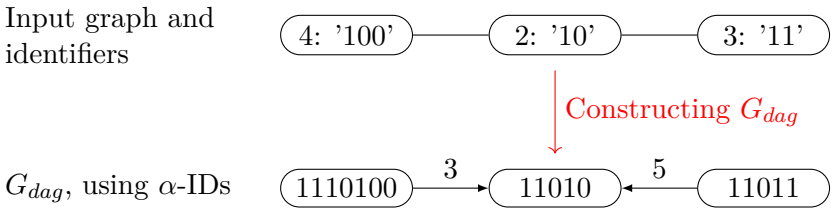


Figure 4.1 – Example of G_{dag} construction

The elimination game is played by the nodes of G_{dag}, round by round, in the following way: on round r, all surviving nodes 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 $2l_{max} + 1$ 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 [36] for a more limited case. Lemmas 3 and 5 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 4. *Let v be any node from G_{dag} not surviving the elimination game. There exists a surviving node u such that $dist(u, v) \leq 2l_{max} + 1$, 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 $2l_{max} + 1$.
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 $l \leq 2l_{max} + 1$. Suppose $l > 2l_{max} + 1$ and focus on $S_{lab}(\Pi)$. Because the edge-labels are integers from $\{1, \ldots, 2l_{max} + 1\}$ and consecutive labels are non equal by Lemma 3, there exists an extremum s_r indexed by $r \in \{2, \ldots, 2l_{max} + 1\}$. 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. □

Lemma 5. *Let $I = \{v \in V \text{ s.t. } survived_v = \texttt{true}\}$ 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 6. *The output set $I = \{v \in V \text{ s.t. } survived_v = \texttt{true}\}$ of RulingSet is a $(2, O(\log N))$-ruling set.*

Proof. Since nodes have unique IDs, I is independent. By Theorem 4 and Lemma 5, I is a $(2, O(\log N))$-ruling set. □

4.3 MIS and Vertex Coloring Algorithms

Let us now present MIS and $(\Delta + 1)$-coloring algorithms 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 [94] . For polylogarithmic Δ, the algorithm 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 algorithms presented here are based on computing and refining defective colorings. Defective colorings were first used to solve the distributed coloring problem in [12] and [75]. Here, we refine the defective coloring differently from the previous works. The exact method is explained below.

4.3.1 Non-uniform Algorithms for MIS and $(\Delta + 1)$-Coloring

The $(\Delta + 1)$-coloring algorithm *DegreeColoring* (Algorithm 2) starts with a Δ-defective coloring *color*, where all nodes start with the same color. Following which, the defect of *color* is reduced iteratively, until the defect is 0 and *color* 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 of *color* 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 (see Figure 4.2). At the start (and end) of each $\mathcal{L}_3$-phase, *color* values are in $\{1, \ldots, \Delta+1\}$. The coloring $\mathcal{L}_2$-phases are used to compute a new coloring *newColor*. The *newColor* values are in $\{1, \ldots, 2\Delta + 2\}$, but the defect of *newColor* is strictly smaller than that of *color*. Following which, a color reduction $\mathcal{L}_2$-phase is executed to reduce the color range of *newColor* to $\{1, \ldots, \Delta+1\}$, and the values of *newColor* are assigned to *color*.

Now we describe the $\mathcal{L}_2$-phases in more detail. In each coloring $\mathcal{L}_2$-phase (see Figure 4.3), nodes with a specific color compute a $(2, O(\log N))$-ruling forest on the subgraph induced by themselves, using *RulingSet* (Algorithm 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

Figure 4.2 – Structure of an $\mathcal{L}_3$-phase in the $(\Delta+1)$-coloring algorithm

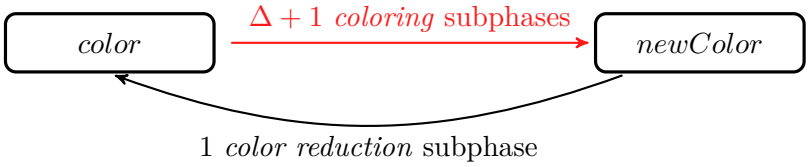
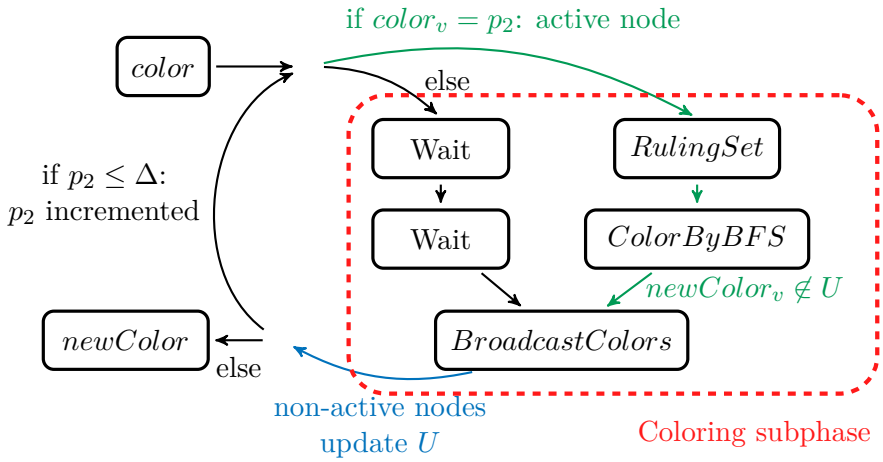


Figure 4.3 – Structure of a coloring $\mathcal{L}_2$-phase p_2 in the $(\Delta+1)$-coloring algorithm



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 that range can increase exponentially if it is not reduced in each $\mathcal{L}_3$-phase.

Coloring $\mathcal{L}_2$-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 $\{1,\ldots,2\Delta+2\}$. 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 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 conveyed through the round number.

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

```
1:  beeping := false                                   ▷ |U| ≤ Δ, U ⊂ {1,...,2Δ+2}.
2:  if inSet then
3:  |   beeping := true                                ▷ Roots initiate BFS
4:  for round r := 1 ; r++ do
5:  |   if beeping then
6:  |   |   Beep          ▷ All nodes beep once. The root is r − 1 hops away.
7:  |   |   newColor := min{k ∈ (ℕ^{>0} \ U) | k ≡ r  mod 2}
8:  |   |   Return newColor                          ▷ newColor ∈ {1,...,2Δ+2}
9:  |   else
10: |   |   Listen
11: |   |   if beep heard then
12: |   |   |   beeping := true
```

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

```
1:  for round r := 1 ; r ≤ 2Δ+2 ; r++ do
2:  |   if newColor = r and changingColor then
3:  |   |   Beep
4:  |   else
5:  |   |   Listen
6:  |   |   if beep heard then
7:  |   |   |   U := U ∪ {r}                         ▷ More unavailable colors
8:  Return U
```

Color Reduction $\mathcal{L}_2$-phase. 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 $\{1,\ldots,\Delta+1\}$ in $\Delta+1$ rounds, after which, nodes with $color = c$ change their color to the smallest available color in $\{1,\ldots,\Delta+1\}$. 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 := ∅                                             ▷ U stores unavailable colors
2:  for round r := 1 ; r ≤ Δ+1 ; r++ do
3:  |   if color = r then
4:  |   |   Beep
5:  |   else
6:  |   |   Listen
7:  |   |   if beep heard then
8:  |   |   |   U := U ∪ {r}
9:  if color = c then
10: |   color := min({1,...,Δ+1} \ U)
11: Return color
```

The $(\Delta+1)$-Coloring Algorithm. In *DegreeColoring* (Algorithm 2), we only require $\mathcal{L}_1$-synchronization points (for *RulingSet* and *ColorByBFS*), introduced in Section 4.1.2. 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 algorithm 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.

Algorithm 2 *DegreeColoring*

IN: *id*: Identifier, Δ: Maximum degree
OUT: *color*: Integer value
$color := 1$
// Each node removes at least one defective edge per $\mathcal{L}_3$-phase
for $\mathcal{L}_3$-phase $p_3 := 1$; $p_3 \leq \Delta$; p_3++ **do**
  $U := \emptyset$ ▷ Stores *newColor* values already chosen during this phase
  $newColor := 0$ ▷ $newColor \in \{1, \ldots, 2\Delta+2\}$ during $\mathcal{L}_3$-phase
  // An $\mathcal{L}_3$-phase starts with $\Delta+1$ coloring $\mathcal{L}_2$-phases
  for coloring $\mathcal{L}_2$-phase $p_2 := 1$; $p_2 \leq \Delta+1$; p_2++ **do**
    if $color = p_2$ **then**
      $inSet := RulingSet(id)$
    $\lightning_1$ ▷ $\mathcal{L}_1$-synchronization point
    if $color = p_2$ **then**
      $newColor := ColorByBFS(inSet, U)$
    $\lightning_1$
    $U := BroadcastColors(newColor, (color = p_2), U, \Delta)$
    $\lightning_2$ ▷ This synchronization point is not needed (strictly explanatory)
  $color := newColor$
  // Followed by one $\mathcal{L}_2$-phase, which contains $\Delta+1$ color reduction $\mathcal{L}_1$-phases
  // Before the $\mathcal{L}_2$-phase, $color \in \{1, \ldots, 2\Delta+2\}$
  for color reduction $\mathcal{L}_1$-phase $p_1 := 1$; $p_1 \leq \Delta+1$; p_1++ **do**
    $color := ColorReduction(color, \Delta, 2\Delta+3-p_1)$
    $\lightning_1$
  // After all color reduction $\mathcal{L}_1$-phases, $color \in \{1, \ldots, \Delta+1\}$
  $\lightning_3$ ▷ Not needed, for clarity only
EndAlgorithm

Lemma 7. *At the start (and end) of an $\mathcal{L}_3$-phase, $color \in \{1, \ldots, \Delta+1\}$.*

Lemma 8. *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 p_3. For any given node v, v has at most d defective edges. It is 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 p_3, 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 $\{1, \ldots, 2\Delta + 2\}$, and at least one edge becomes non-defective. □

Theorem 9. *Algorithm 2 solves* $(\Delta + 1)$*-coloring in* $O(\Delta^2 \log n + \Delta^3)$ *rounds.*

Proof. Algorithm 2 is correct after Δ $\mathcal{L}_3$-phases, by Lemmas 7 and 8.

As for the time complexity of Algorithm 2, both *RulingSet* and $ColorByBFS$ take $O(\log n)$ rounds, whereas $BroadcastColors$ takes $O(\Delta)$ rounds. As a result, each coloring $\mathcal{L}_2$-phase takes $O(\Delta + \log n)$ rounds. Moreover, each color reduction $\mathcal{L}_2$-phase takes $O(\Delta)$ rounds. Finally, Algorithm 2 terminates after $O(\Delta^2)$ coloring $\mathcal{L}_2$-phases and $O(\Delta^2)$ color reduction $\mathcal{L}_2$-phases. □

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.

4.3.2 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.

Uniform Functions. 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.

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 $DegreeColoring$.

function UnifBroadcastColors(*newColor*, *changingColor*, U): U

```
1:  for L1-phase p1 := 1 ; p1++ do                ▷ L1-phase consists of two rounds
2:      // First round
3:      if newColor ≥ p1 and changingColor then
4:          BEEP                                   ▷ Not finished yet
5:      else
6:          LISTEN
7:          if no beep heard then
8:              Return U                           ▷ If all neighbors beeped their colors
9:      // Second round
10:     if newColor = p1 and changingColor then
11:         BEEP                                   ▷ Communicate your color
12:     else
13:         LISTEN
14:         if beep heard then
15:             U := U ∪ {p1}                      ▷ Keep neighbors' newColor values
```

$UnifColorReduction$ 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 conveys its *color* value to its neighbors by beeping in the first round of the $\mathcal{L}_1$-phase indexed by $color_v$. Nodes with the highest color in their neighborhood choose the smallest available color (colors previously conveyed 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 $\{1, \ldots, \Delta + 1\}$). 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.

Local Termination Component. 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

function UnifColorReduction(*color*, *U*): *color*, *sameColor*

```
1:  sameColor := false
2:  for L1-phase p1 := 1 ; p1++ do                ▷ L1-phase consists of two rounds
3:      // First round
4:      if color = p1 then
5:          Beep
6:      else
7:          Listen
8:          if beep heard then
9:              U := U ∪ {p1}
10:     // Second round
11:     if color > p1 then
12:         Beep
13:     else
14:         // Only a node with the highest color in its neighborhood hears no beep
15:         Listen
16:         if beep heard then
17:             Return (color, sameColor)
18:         else
19:             color := min({1, ..., p1} \ U)
20:             if color = p1 then
21:                 sameColor := true
22:             Return (color, sameColor)
```

function ReduceColors(*color*, *U*): *color*

```
1: finished := false
2: while not finished do                          ▷ At most c L3-phases
3:     (color, finished) := UnifColorReduction(color, U)
4:     ↯2
5:     U := UnifBroadcastColors(color, finished, U)
6:     ↯2                          ▷ Actually, also an L3-synchronization point
7: Return color
```

same color. Then, the executing node can exit the outermost loop and thus locally terminate the algorithm (see lines 27 to 33).

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 p_2, nodes with color p_2 check for a collision by using *UnifCollisionBeep*. If no neighboring node with the same color p_2 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

function ColorCollision(*id*, *color*): *collision*

```
1: for L2-phase p2 := 1 ; p2++ do                    ▷ At most Δ + 1 L2-phases
2:     if color = p2 then
3:         collision := UnifCollisionBeep(id)
4:         Return collision
5:     ↯2
```

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 p_1^{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 p_1 is greater than the length of the α-ID.

function UnifCollisionBeep(*id*): *collision*

```
1: collision := false
2: for L1-phase p1 := 1 ; p1++ do          ▷ L1-phase consists of two rounds
3:     if p1 > |α(id)| then
4:         Return collision
5:     if α(id)[p1] = 0 then
6:         BEEP ; LISTEN
7:         if beep heard in the second round then
8:             collision := true
9:     else
10:        LISTEN ; BEEP
11:        if beep heard in the first round then
12:            collision := true
```

The Uniform $(\Delta+1)$-Coloring Algorithm. 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 chosen a new color. The algorithm'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 algorithm.

Theorem 10. *MIS and $(\Delta+1)$-coloring can be solved in $O(\Delta^2 \log n + \Delta^3)$ rounds with an algorithm uniform in both Δ and N.*

Algorithm 3 *UnifDegreeColoring*

```
1: IN: id: Identifier   OUT: color: Integer value
2: U_t := ∅          ▷ Stores color values chosen for output by terminated neighbors
3: color := 1
4: // Each node removes at least one defective edge per L_5-phase
5: for L_5-phase p_5 := 1 ; p_5++ do
6:     U := ∅          ▷ Stores newColor values already chosen during this phase
7:     newColor := 0          ▷ newColor ∈ {1, ..., 2Δ + 2} during the L_5-phase
8:     // This L_3 loop is an L_4-phase
9:     for coloring L_3-phase p_3 := 1 ; p_3++ do
10:        if color = p_3 then
11:            inSet := RulingSet(id)
12:        ↯_1
13:        if color = p_3 then
14:            newColor := ColorByBFS(inSet, U ∪ U_t)
15:        ↯_1
16:        U := UnifBroadcastColors(newColor, color = p_3, U)
17:        ↯_2   ▷ L_2-synchronization point here, thus we have coloring L_3-phases
18:        // From here on, local termination component for L_4-phase
19:        if color > p_3 then
20:            BEEP                                  ▷ Beep if new color still not chosen
21:        else
22:            LISTEN          ▷ Check if any neighbor is still choosing a new color
23:            if no beep heard then Exit L_3 loop
24:    ↯_4       ▷ Crucial because some nodes end the L_4-phase earlier than others
25:    color := ReduceColors(newColor, U_t)          ▷ After, color ∈ {1, ..., Δ + 1}
26:    ↯_4
27:    // From here on, local termination component for L_5 loop
28:    collision := ColorCollision(color)
29:    ↯_3                                  ▷ Because ColorCollision is an L_3-phase
30:    U_t := UnifBroadcastColors(color, collision = false, U_t)
31:    ↯_2                                  ▷ Because UnifBroadcastColors is an L_2-phase
32:    if not collision then
33:        EndAlgorithm                                          ▷ Exit L_5 loop
```

4.4 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, Algorithm 3 is less efficient. Using ideas from [63] and [11], it is possible to design a $(\Delta+1)$-coloring algorithm 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 11. *MIS and $(\Delta+1)$-coloring can be solved with $O(a^2 \log^2 n + a^3 \log n)$ round complexity in $\mathcal{BEEP}_S$, where a is the arboricity of the communication graph.*

To support this theorem, we design two coloring algorithms 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 an algorithm uniform in a, since a may be harder to obtain than an upper bound on N. The following results from [11] are used to obtain the following algorithms.

Lemma 12. *[11] If G is of arboricity a, at least $\frac{\epsilon}{2+\epsilon}|V|$ nodes have a degree less than $(2+\epsilon)a$.*

Theorem 13. *[11] 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 algorithms. 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 $\{1, \ldots, \Delta_a + 1\}$, due to their high degree, and end the function uncolored, represented by the color 0. We use *LimitedColorReduction*, a slightly modified version of *ColorReduction*, presented below. The only change is that *color* is set to 0 if the set of available colors A is an empty set (line 13).

function LimitedColorReduction(*color*, c_1, c_2): *color*

```
1:  U := ∅                                          ▷ U stores unavailable colors
2:  for round r := 1 ; r ≤ c1 + 1 ; r++ do
3:      if color = r then
4:          BEEP
5:      else
6:          LISTEN
7:          if beep heard then
8:              U := U ∪ {r}
9:  A := {1, ..., c1 + 1} \ U                       ▷ Set of available colors
10: if color = c2 and A ≠ ∅ then
11:     color := min(A)
12: else if color = c2 and A = ∅ then
13:     color := 0
14: Return color
```

Lemma 14. *Let $\Delta_a = (2+\epsilon) \cdot a$, with $\epsilon > 0$. Given the input $c = \Delta_a$, LimitedDegreeColoring outputs a (Δ_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 $O(a^2 \cdot \log n + a^3)$.*

function LimitedDegreeColoring(id, c): *color*

```
1:  color := 1
2:  for L3-phase p3 := 1 ; p3 ≤ c ; p3++ do
3:      U := ∅                ▷ Stores newColor values already chosen during this phase
4:      newColor := 0
5:      for L2-phase p2 := 1 ; p2 ≤ c + 1 ; p2++ do
6:          if color = p2 then
7:              inSet := RulingSet(id)
8:          ↯1                                        ▷ L1-synchronization point
9:          if color = p2 then
10:             newColor := ColorByBFS(inSet, U)
11:             if color ∉ {1, ..., 2c + 2} then
12:                 Return 0                          ▷ Not a good color
13:         ↯1
14:         U := BroadcastColors(newColor, (color = p2), U, c)
15:     color := newColor
16:     for L1-phase p1 := 1 ; p1 ≤ c + 1 ; p1++ do
17:         color := LimitedColorReduction(color, c, 2c + 3 − p1)
18:         if color = 0 then
19:             Return 0          ▷ No color in {1, ..., c + 1} could be chosen
20: collision := ColorCollision(color)
21: if collision then
22:     Return 0                  ▷ If not properly colored, no color is chosen
23: Return color
```

Proof. The round complexity is straightforward.
LimitedDegreeColoring outputs a (Δ_a+1)-coloring on the subgraph of nodes with non-zero colors because all colors are chosen from $\{1,\ldots,2\Delta_a+2\}$, if available, and are then reduced to $\{1,\ldots,\Delta_a+1\}$. *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 $color_u = color_v$. □

4.4.1 $(\Delta+1)$-Coloring Uniform in N

First, let us focus on the *first algorithm*, uniform in N. *LimitedDegreeColoring* is executed iteratively by uncolored nodes until all nodes are properly colored. Since a is known, and by Lemma 12, 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 13, executing

LimitedDegreeColoring $l = O(\log N)$ times 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 Algorithm 3. This takes an additional $O(a^2 \cdot \log^2 n)$ rounds.

Algorithm 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 $p_4 = 1$; p_4++ **do** ▷ At most $l = \frac{2}{\epsilon} \cdot \log n$ $\mathcal{L}_4$-phases
    $color := LimitedDegreeColoring(id, \Delta_a)$
    $\lightning_4$
    if $color \neq 0$ **then**
        $color := color + (p_4 - 1) \cdot (\Delta_a + 1)$
        Exit $\mathcal{L}_4$ loop
$\lightning_5$ ▷ *color* is an $O(a \cdot \log n)$-coloring
$color := ReduceColors(color, \emptyset)$ ▷ At most $O(a^2 \cdot \log^2 n)$ rounds
EndAlgorithm ▷ $color \in \{1, \ldots, \Delta + 1\}$

Theorem 15. *Algorithm 4 solves MIS and $(\Delta + 1)$-coloring with $O(a^2 \log^2 n + a^3 \log n)$ round complexity. This algorithm 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 Algorithm 4, only uncolored nodes (V_{rem}) participate in *LimitedDegreeColoring*. Since the subgraph induced by V_{rem} also has arboricity at most a, by Lemmas 12 and 14, $\frac{\epsilon}{2+\epsilon}|V_{rem}|$ 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 13, 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 14. □

4.4.2 $(\Delta + 1)$-Coloring Uniform in a

In the *second algorithm* (uniform in a), we compute an upper bound on a. This is done by estimating a iteratively. At each iteration ($\mathcal{L}_5$-phase) p_5, a is estimated to be 2^{p_5} 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 algorithm.

Theorem 16. *Algorithm 5 solves MIS and $(\Delta + 1)$-coloring with $O(a^2 \log^2 n + a^3 \log n)$ round complexity. This algorithm is uniform in arboricity a but non-uniform in N.*

Algorithm 5 *UnifAArbColoring*

1: **IN:** *id*: Identifier, N: Polynomial upper bound on n, ϵ: Parameter
2: **OUT:** *color*: Integer value
3: **for** $\mathcal{L}_5$-phase $p_5 := 1$; p_5++ **do** ▷ At most $1 + \lfloor \log a \rfloor$ $\mathcal{L}_5$-phases.
4: $\quad \Delta_{p_5} := (2+\epsilon) \cdot 2^{p_5}$
5: $\quad len := \frac{2}{\epsilon} \cdot \log N$
6: $\quad$ **for** $\mathcal{L}_4$-phase $p_4 := 1$; $p_4 \leq len$; p_4++ **do**
7: $\quad\quad color := LimitedDegreeColoring(id, \Delta_{p_5})$
8: $\quad\quad \lightning_4$
9: $\quad\quad$ **if** $color \neq 0$ **then**
10: $\quad\quad\quad color \mathrel{+}= (\Delta_{p_5} - 2 - \epsilon + p_5 - 1) \cdot len + (p_4 - 1) \cdot (\Delta_{p_5} + 1)$
11: $\quad\quad\quad$ Exit $\mathcal{L}_5$ loop
12: $\quad \lightning_5$ ▷ Not needed, for clarity only
13: $\lightning_6$ ▷ color is an $O(a \cdot \log n)$-coloring
14: $color := ReduceColors(color)$ ▷ At most $O(a^2 \cdot \log^2 N)$ rounds
15: EndAlgorithm ▷ $color \in \{1, \ldots, \Delta + 1\}$

Proof. Let us first prove that Algorithm 5 solves $(\Delta + 1)$-coloring. At the end of $\mathcal{L}_5$-phase p_5, by Lemma 14, all nodes with degree less than $\Delta_{p_5} = (2+\epsilon) \cdot 2^{p_5}$ are colored. By Lemmas 12 and 14, and Theorem 13, at $\mathcal{L}_5$-phase $p_5 = 1 + \lfloor \log a \rfloor$, $\Delta_{p_5} \geq (2+\epsilon) \cdot a$ 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. □

4.5 Uniform Algorithms for the 2-hop Variants

To obtain algorithms for 2-hop MIS and 2-hop coloring, we provide and use a general transformer, the *SquareSim* algorithm (Algorithm 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 an algorithm 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 17. *An algorithm 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 17 to the previous algorithms, we obtain algorithms for solving 2-hop coloring with $(\Delta^2 + 1)$ colors and 2-hop MIS. These algorithms are very efficient on

Algorithm 6 Simulating the square communication graph: *SquareSim*

```
1: IN: beep: Boolean value  OUT: detectedBeep: Boolean value
2: detectedBeep := false
3: if beep then
4: |   Beep                          ▷ Beep to neighbor nodes: First round
5: else
6: |   Listen
7: |   if beep heard then
8: |   |   detectedBeep := true
9:
10: if detectedBeep then
11: |   Beep                  ▷ Relay beep to distance 2 nodes: Second round
12: else
13: |   Listen
14: |   if beep heard and not beep then
15: |   |   detectedBeep := true
16: EndAlgorithm
```

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 interference. In the next section, we show how this can be used to simulate the stronger $\mathcal{CONGEST}$ communication model and obtain an $O(a)$-coloring.

Corollary 18. *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 [5], showing that $a(G^2) \leq 2^3 \cdot a \cdot \Delta$, we obtain Corollary 19, which provides a more efficient result for graphs with small arboricity.

Corollary 19. *2-hop MIS and 2-hop* $(\Delta^2 + 1)$*-coloring are solved by the two algorithms in Sect. 4.4 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*.*

4.6 $\mathcal{CONGEST}$ 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 $\mathcal{CONGEST}$ 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* (Algorithm 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*.

Algorithm 7 *InitCongest*

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

Our simulation algorithm *SimCongest* (Algorithm 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.

The following lemma is straightforward.

Lemma 20. *Given a 2-hop c-coloring, the $\mathcal{CONGEST}$ model with edge bandwidth B can be simulated in $\mathcal{BEEP}_S$, with an $O(c^2 \cdot B)$ multiplicative factor.*

An $O(a)$-Coloring Algorithm in the Beeping Model. Finally, using the simulation of $\mathcal{CONGEST}$, one can use the result of Barenboim and Elkin [13] (given for $\mathcal{CONGEST}$), to obtain an $O(a)$-coloring in $\mathcal{BEEP}_S$.
It is done by first computing, in $\mathcal{BEEP}_S$, a 2-hop (Δ^2+1)-coloring in $O(a^2\Delta^2 \log^2 n + a^3\Delta^3 \log n)$ rounds (Corollary 19). Then the $O(a)$-coloring from [13] (with $O(a^\mu \log n)$ round complexity) is combined with the $\mathcal{CONGEST}$ simulation, using the (Δ^2+1)-coloring obtained before. By Lemma 20, the resulting simulation of the $O(a)$-coloring algorithm 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 algorithm in $\mathcal{BEEP}_S$. Notice that now by using this coloring algorithm, together with the *SquareSim* algorithm, to obtain a 2-hop $O(a \cdot \Delta)$-coloring (see Section 4.5), we reduce the time multiplicative factor when simulating $\mathcal{CONGEST}$ algorithms. Consequently, one obtains a more efficient simulation.

Algorithm 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 L_3-phase p_3 := 1 ; p_3 ≤ c ; p_3++ do
4:     for L_2-phase p_2 := 1 ; p_2 ≤ c ; p_2++ do
5:         for L_1-phase p_1 := 1 ; p_1 ≤ B ; p_1++ do
6:             if p_3 = color then                          ▷ p_3 can send its p_1th bit to p_2
7:                 if p_2 ∈ Nb and mSend(p_2)[p_1] = 0 then
8:                     // Send a 0 message.
9:                     Beep ; Listen
10:                else if p_2 ∈ Nb and mSend(p_2)[p_1] = 1 then
11:                    // Send a 1 message.
12:                    Listen ; Beep
13:                else
14:                    Listen ; Listen                      ▷ Synchronize
15:            else if p_2 = color then
16:                // Listen for a possible incoming p_1th bit.
17:                Listen ; Listen
18:                // Then append the received bit in mRec.
19:                if beep heard in first round then
20:                    mRec(p_3) := mRec(p_3) ‖ 0
21:                if beep heard in second round then
22:                    mRec(p_3) := mRec(p_3) ‖ 1
23:            else
24:                Listen ; Listen                          ▷ Synchronize
25: EndAlgorithm
```

4.7 Extended Balanced Execution Technique (EBET)

We remind that the "Balanced Execution Technique" (BET) from Förster *et al.* [58] 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 algorithms in $\mathcal{BEEP}_S$.

4.7.1 Introducing EBET

Synchronization points are not a natural primitive in $\mathcal{BEEP}_S$: 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 algorithms are difficult to design in a uniform manner without the use of synchronization points. Therefore, we want to be able to design an algorithm $\mathcal{P}$ using synchronization points, and then apply a “technique” on the formal description of $\mathcal{P}$, so that the result is an algorithm that can be run in

$\mathcal{BEEP}_S$ (not necessarily a formal description). The resulting algorithm is called $\mathcal{P}_{sim}$. The technique we use for that is EBET.

Extended Balanced Execution Technique. 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 $\mathcal{BEEP}_S$. 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 $\mathcal{P}$ (with a similar round complexity), and proving this is the main focus of Section 4.7.3.

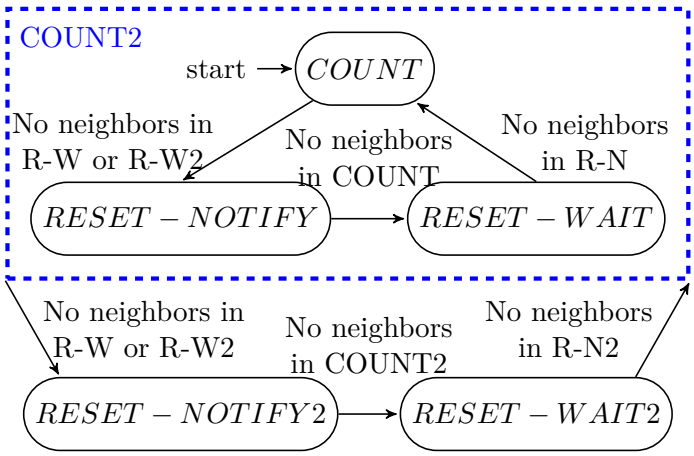
Outline. In the first section (Section 4.7.2), we describe the balanced counter technique (extending that of [58]), which allows EBET to maintain a balanced round counter, and to guarantee the synchronization property for all $\mathcal{L}_i$-phases. When abstracted to a higher level, the synchronization property results in the simulation of $\mathcal{L}_i$-synchronization points. In the second section (Section 4.7.3), 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.

4.7.2 Extending the Balanced Counter Technique for EBET

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 4.4 (CT, R-N, R-W, R-$N2$ and R-$W2$). A node v with $state(v, p) = CT$ 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).

Figure 4.4 – Finite State Machine Component for EBET ($k = 2$)



Each simulation $\mathcal{L}_1$-phase contains exactly 8 slots and is used to convey 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 convey the counter value (rC) modulo 3, and the other slots (indexed 3 to 7) are used to convey the current FSM state of a node ($state$). For any given node v, the information is conveyed 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 $state = R\text{-}N$, then v beeps in slot 4,
- If $state = R\text{-}W$, then v beeps in slots rC mod 3 and 5,
- If $state = R\text{-}N2$, then v beeps in slot 6,
- If $state = R\text{-}W2$, 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 4.4). These conditions are essential for the *balanced counter* and *synchronization* properties (Lemmas 24 and 25) in EBET. For any given node v, the allowed state transitions are:

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

5. *R-N2* $\rightarrow$ *R-W2* if no node $u \in \mathcal{N}(v)$ is in $CT2$,

6. *R-W2* $\rightarrow CT$ 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 $1 \rightarrow 2 \rightarrow 3$) as an $\mathcal{L}_1$-*cycle*, and the second cycle (transitions $4 \rightarrow 5 \rightarrow 6$) 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 $CT \rightarrow$ *R-N* $\rightarrow$ *R-W* $\rightarrow CT$. *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 conveyed: 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 conveyed, which stalls the *increment* function of these neighboring nodes (see paragraph "Functions of the Balanced Counter Technique"). As such, the neighboring rC values satisfy $rC \leq 1$ (Lemma 23). 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 $CT \rightarrow$ *R-N2* $\rightarrow$ *R-W2* $\rightarrow CT$. 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-N2* state).

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*, *reset*2 and *increment*. These functions can only be invoked by participating nodes and increment p_1, p_2 and rC while ensuring the synchronization and balancing properties. When node v increments p_1 (resp., p_2), that means that v has done a full $\mathcal{L}_1$-cycle (resp., $\mathcal{L}_2$-cycle). Consequently, p_1 (resp., p_2) 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 p_1 and p_2 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 $rC(v) - 1 \bmod 3$. 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, *reset*2 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. *reset*2) 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 Section 4.7.3.

Properties of the Balanced Counter Technique. First, we give a few lemmas (Lemmas 21, 22 and 23), which are then used to prove both the balancing and synchronization properties (Lemmas 24 and 25).

Lemma 21. *For any given simulation $\mathcal{L}_1$-phase p and node v, if $state(v, p) =$ R-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 $u \in \mathcal{N}(v)$ $state(u, p) \notin \{R\text{-}N, R\text{-}W\}$. □

By proving Lemma 21, we also prove its contrapositive - Lemma 22 - which is used to simplify the proof of Lemma 23. Indeed, Lemma 22 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 22. *For any given simulation $\mathcal{L}_1$-phase p and node v, if there exists $u \in \mathcal{N}(v)$ such that $state(u, p) \in \{R\text{-}N, R\text{-}W\}$, then $state(v, p) \neq R\text{-}W2$.*

Lemma 23. *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 22, $state(u, p-1) \neq R\text{-}W2$. 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 21, 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 23, 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 24).

Lemma 24 (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-W* or *R-W*2 in $p-1$ (node u cannot transition from *R-N* to *CT* in a single phase). Thus, by Lemma 23, $rC(v,p-1) \leq 1$. 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 25), 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 25 (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-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-W*2 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-N*2 (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 22, $state(v, p-1) \neq R\text{-}W2$. 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 21, $state(v, p-1) \neq R\text{-}W$. 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 $reset2$ (p_1 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.

4.7.3 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.

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 $next(v, p) = \texttt{false}$, 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 $r \bmod 3 + 8$ 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.

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 26. *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 27). 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 28 is straightforward.

Lemma 27. *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, $next(v,p)=\texttt{true}$ thus $p'<p$ (Lemma 26).
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 $next(v,p)=\texttt{false}$. 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 28. *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 29 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 29. *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 $R_{sim} = O(R)$. □

4.8 Summary

A brief description of this chapter follows.

- First, in Section 4.2, we present a $(2, O(\log N))$-ruling set algorithm and introduce a tool for analyzing competition algorithms: the *labeled "deterministic"* competition graphs. Using such graphs in algorithm analysis is inspired by [66]. Here we adapt this technique to the case of deterministic competition algorithms. In particular, the ruling set algorithm is a deterministic competition algorithm and its analysis rely on the labeled deterministic competition graphs. This general tool may be useful also in future studies of MIS and coloring.
- Then, we present a series of *uniform* MIS and coloring algorithms in Sections 4.3 through 4.5.
  - First, we present $O(\Delta^2 \log n + \Delta^3)$ deterministic uniform algorithms for MIS and $(\Delta + 1)$-coloring, where Δ is the unknown maximum degree (Section 4.3). These algorithms 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 with regards to n, which is very efficient.
  - Then, we extend the previous algorithms with time complexity dependent on Δ, to algorithms with time complexity dependent on the arboricity a (Section 4.4). We get $O(a^2 \log^2 n + a^3 \log n)$ time MIS and $(\Delta+1)$-coloring uniform algorithms. This results in efficient polylogarithmic time complexity for the large family of graphs where $a = O(\log^c n)$.
  - Finally, we extend the previous algorithms into $O(a^2\Delta^2 \log^2 n + a^3\Delta^3 \log n)$ time 2-hop coloring and 2-hop MIS uniform algorithms (Section 4.5).
- In Section 4.6, given a 2-hop coloring, we prove that the $\mathcal{CONGEST}$ model can be simulated with an $O(\Delta^4)$ multiplicative overhead. Using this simulation and the algorithm proposed in [13] for the $\mathcal{CONGEST}$ 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 algorithm in $\mathcal{BEEP}_S$, for any given positive constant $\mu < 1$. To the best of our knowledge, this is the first coloring algorithm using less than $\Delta + 1$ colors in $\mathcal{BEEP}_S$.
- Finally, in Section 4.7, we prove that the Balanced Execution Technique can be extended 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 algorithms in $\mathcal{BEEP}_S$. In particular, all uniform vertex coloring algorithms presented in this chapter cannot be implemented with $\mathcal{L}_1$-synchronization points only.

Chapter 5

Optimal Leader Election Algorithm

In this chapter, we are interested in interference control on a global scale (in the synchronous starts setting). For that reason, we consider *leader election* which is a global scale symmetry-breaking problem. Once a leader is elected, it can force network-wide coordination in order to avoid or take advantage of collisions. This is illustrated in the next chapter, where the leader is essential in implementing the multi-broadcast primitive.

Traditionally, leader election solutions rely on relatively large messages (i.e., of $O(\log n)$ size). Recently, however, [25] assumed only 1-bit messages and proposed the first optimal ($O(D+\log n)$ rounds) deterministic leader election solution in this setting. Compared to 1-bit messages in a model without interference, as considered in [25], beeps appear to be a weaker communication mechanism. Therefore, a natural question is whether a time-optimal ($O(D + \log n)$ rounds) deterministic leader election solution can also be obtained in the beeping model. We answer this question positively in this chapter. To do so, we do not use local symmetry-breaking primitives to avoid collisions and ensure that nodes can communicate large messages (i.e., IDs) reliably to their neighbors. Instead, nodes exchange a small amount of collision-tolerant information, by utilizing the *strong degree of synchronization* (global clocks) and non-destructive interference, provided by the model. More precisely, when nodes hear a beep (possibly a collision) in some round r, the information received is the round number r[1]. This technique allows nodes to efficiently compare IDs in a pipeline-like manner.

5.1 Introduction

The leader election (LE) problem, where a single (leader) node is given a distinguished role in the network, is a fundamental building block in algorithm design. This is because a leader can initiate and coordinate behaviors in the network, often making leader election a crucial first step in applications requiring communication and agreement on a global scale. For example, more advanced communication primitives such as broadcast, gossiping and multi-broadcast, rely on a leader to coordinate transmissions [40] (see also Chapter 6).

Being a fundamental coordination problem, leader election has received a lot of attention (see Section 5.1.1) in the beeping model. Probabilistic and deterministic

[1]Notice that some degree of synchronization between neighboring nodes is crucial for this to work. Indeed, this technique does not work in the uncoordinated starts setting.

(non-optimal) solutions were proposed for general graphs, and a time complexity lower bound of $\Omega(D + \log n)$ was established (D is the diameter of the network, and n its size). We show in this work that an asymptotically time-optimal deterministic algorithm can be designed. This algorithm gives rise to an anonymous (not using IDs) randomized algorithm that also matches the lower bound (Section 5.4.1).

Deterministic Uniform Leader Election and α-identifiers. We remind that the α-encoding [25] of an integer $i \in \mathbb{N}^{>0}$ is a word obtained from the binary representation bin of i. By definition, $\alpha(i) = 1^{|bin|} \parallel 0 \parallel bin$ (see Section 2.2). In the leader election solution presented in this chapter, instead of IDs, nodes compare their α-encodings (α-IDs). As a result, the leader election solution compares binary words instead of integer values. Importantly, a word x is *well-formed* if there exists an integer i such that $x = \alpha(i)$. This integer can be obtained by taking the last $\lfloor \frac{|x|}{2} \rfloor$ bits of x. Additionally, it is simple to prove that for every word x, there is at most one such integer i. Thus the α^{-1} function (α's "inverse") is defined on well-formed words.

5.1.1 Specific Related Work

Leader election (LE), being a fundamental problem in distributed computing, has been studied in various models. In particular, recent models designed for wireless networks assume that simultaneous communications interfere with each other. Consequently, leader coordination is even more important in these models.

Even though computational complexities (in particular time complexity) for LE are key aspects in the algorithmic design, additional properties are also of concern: for example, one might want nodes to detect termination, or to ensure that there is never more than one leader node during any execution (*safety property*).

Related Work in $\mathcal{BEEP}_S$ Ghaffari and Haeupler [60] present the first LE algorithm for $\mathcal{BEEP}_S$, which elects a leader in $O(D + \log n) \cdot O(\log^2 \log n)$ rounds w.h.p. [60] also gives a lower bound of $\Omega(D + \log n)$ rounds for LE, applicable both to deterministic and randomized (w.h.p. time) algorithms. This bound can be compared to the $\Omega(D)$ lower bound in the $\mathcal{CONGEST}$ model [76]. $\mathcal{CONGEST}$ differs from $\mathcal{BEEP}_S$ in that any given node can send (different) messages of $O(\log n)$ bits to each of its neighbors during a round. When nodes receive messages, there are no collisions and they can distinguish from which edge they received a particular message. Intuitively, since a beep can convey at most one bit, additional $\Omega(\log n)$ rounds are necessary [83, 25, 47]. Following the result from [60], Czumaj and Davies [39] presented a randomized LE algorithm with $O(D + \log n)$ expected time in $\mathcal{BEEP}_S$. In both randomized algorithms, the safety property is guaranteed w.h.p., but some upper bound N on the number of nodes n is required. As for deterministic LE, Förster et al. [58] give the first algorithm in $\mathcal{BEEP}_S$, with an $O(D \cdot \log n)$ round complexity. This algorithm is uniform in both n and D. The round complexities of different LE algorithms, including those presented in this work, are compared below (see Table 5.1).

We remind that upper bounds in $\mathcal{BEEP}_S$ apply to the radio network model with $O(\log n)$ bit messages and collision detection (since algorithms designed in $\mathcal{BEEP}_S$ can be straightforwardly translated to this model). For instance, [60, 58] give the best randomized (w.h.p.) and deterministic results for the radio network model prior to our work.
Importantly, for both models, these previous results are not tight, especially for deterministic leader election.

Table 5.1 – LE algorithms in the beeping model

Reference	Round complexity	Safety	Knowledge
[60]	$O(D + \log n) \cdot O(\log^2 \log n)$ w.h.p.	w.h.p.	$N = n^c$
[58]	$O(D \cdot \log n)$ deterministic time	Deterministic	None
[39]	$O(D + \log n)$ expected time	w.h.p.	$N = n^c$
Our work	$O(D + \log n)$ deterministic time	Deterministic	None
Our work	$O(D + \log n)$ w.h.p.	w.h.p.	$N = n^c$

Related Work in $\mathcal{CONGEST}$ with 1-bit Messages. Amongst the extensive leader election literature in other models, Casteigts et al. [25] is particularly relevant to this work. [25] proposes an $O(D + \log n)$ time deterministic LE algorithm in the constant-size $\mathcal{CONGEST}$ model, where the algorithm is uniform in both the number of nodes n and the diameter D. This model is much stronger than $\mathcal{BEEP}_S$, in that a node can easily learn its local topology and has direct links to communicate with its neighbors, whereas the absence of such links in the beeping model causes interference and makes directed messages (with known sender and receiver) unachievable or plainly inefficient. Notice that by using a 2-hop coloring and by separating in time the transmission of messages according to the colors of both the sender and receiver (as shown in Section 4.6), the constant-size $\mathcal{CONGEST}$ model can be simulated, but with a prohibitive multiplicative factor of $O(\Delta^4)$ (where Δ is the maximum degree).
Nevertheless, one of the main contributions of [25] is a rooted (in the maximum ID node) spanning tree construction and an *information diffusion* algorithm, designed to spread the maximum identifier efficiently, in a pipeline-like manner (rather than performing consecutive local comparisons on complete identifiers). This latter shift is crucial to the time-optimality of their algorithm, and is used here to improve on the $O(D \cdot \log n)$ result from [58].

5.2 Uniform Eventual Leader Election

The algorithm (Algorithm 9) is described first (Section 5.2.1). Then, in Section 5.2.2, k-balanced messages are presented. They are used to allow constant-size *communication phases* composed of rounds and dedicated to the communication of (large) messages respecting local constraints. Using the k-balanced message technique, a detailed description of the communication phases (appearing in Algorithm

9) is given in Section 5.2.3. Finally, in Section 5.2.4, we relate the presented techniques to existing works in $\mathcal{CONGEST}$ models.

5.2.1 Description

Algorithm 9 Uniform Eventual Leader Election Algorithm

1: **IN:** *id*: identifier
2: **OUT:** *leader*: boolean, *leaderId*: identifier
3: $candidate :=$ `true`, $prefix := \epsilon$, $suspicious :=$ `false` ▷ ϵ is the empty word
4: $leaderId := 0$, $leader :=$ `false` ▷ *id* and $leaderId$ are IDs, from $\{1, \ldots, N\}$
5: **for** diffusion phase $p := 1$; p++ **do**
6: | *// First, a communication phase with c rounds.*
7: | Communicate $(prefix, suspicious)$ to all neighboring nodes.
8: | *// Then, apply predicates of rules 1 to 5 on received*
9: | *// $(prefix, suspicious)$ pairs.*
10: | Use received $(prefix, suspicious)$ pairs to update $prefix$, *candidate* and
11: | *suspicious*
12: | **if** not *candidate* **then**
13: | | $leader :=$ `false`
14: | **else if** $prefix = \alpha(id)$ **then**
15: | | $leader :=$ `true`
16: | **if** $prefix$ is well-formed **then**
17: | | $leaderId := \alpha^{-1}(prefix)$

All nodes aim to spread their α-ID ($\alpha(id)$ in Algorithm 9) to the whole network (*information diffusion algorithm*). They execute loosely synchronized bit-wise comparisons and propagate the bits of the highest detected prefix (of α-ID). All nodes start out as *candidates*, with two variables: $prefix$ and *suspicious*. The binary word $prefix$ is initialized to the empty word ϵ and represents the prefix of an α-ID. Most of the time, it represents the highest prefix of which the node is aware. Each node adapts its $prefix$ by adding or removing the less significant bits, depending on the information gathered. The boolean *suspicious* is initialized to `false` and indicates whether the node removed bits from $prefix$ in the last phase.

Nodes execute *diffusion phases* (of c rounds each) synchronously. A diffusion phase consists of one *communication phase* of $c = O(1)$ rounds (line 7), used to send $prefix$ and *suspicious* to all neighbors, followed by a (limited) modification of $prefix$. Depending on the newly computed $prefix$ value, nodes decide on the outputs of *leader* and $leaderId$.

Communication Phase. The communication phase is described in detail in Section 5.2.3. In the same phase, each node receives $(prefix, suspicious)$ pairs from its neighbors, but does not know which node sent which message, nor how many nodes sent any of these messages (*multiplicity*).

Limited Modification of $prefix$. After the communication phase, any node v checks if $prefix_v$ is a *locally greater prefix*, using the received pairs (see details below) and the previously gathered information. If this is the case, it appends a bit from its α-ID to $prefix_v$ (if $prefix_v$ is a proper prefix of $\alpha(id_v)$), or does nothing (if $prefix_v = \alpha(id_v)$). Otherwise, it modifies $prefix_v$ depending on the highest detected $prefix$ value, and becomes a *follower*. It can no longer become a leader. If that modification removes bits from $prefix_v$, node v is said to be *suspicious* for the following phase, and $suspicious_v$ is assigned to `true` for one phase.

The five rules below, inspired from [25], associate conditions (predicates) to actions. A predicate evaluated to `true` triggers the associated action. In line 10, these predicates are evaluated (by some node v) on the set of the received $(prefix, suspicious)$ pairs, in the given order of priority, and the first triggered action is performed.

1. If there exists a suspicious neighbor u, such that $prefix_u$ is a proper prefix of $prefix_v$, remove $min\{|prefix_v| - |prefix_u|, 3\}$ letters from the end of $prefix_v$.

2. If $prefix_v = (z \parallel 0 \parallel w)$ with $w \neq \epsilon$ and there exists a neighbor u with $prefix_u = (z \parallel 1 \parallel y)$, delete $|w|$ letters from the end of $prefix_v$.

3. If $prefix_v = (z \parallel 0)$ and there exists a neighbor u with $prefix_u = (z \parallel 1 \parallel y)$, then change $prefix_v$ to $(z \parallel 1)$.

4. If there exists a neighbor u with $prefix_u = (prefix_v \parallel 1 \parallel w)$ then append 1 to $prefix_v$.

5. If there exists a neighbor u with $prefix_u = (prefix_v \parallel 0 \parallel w)$ then append 0 to $prefix_v$.

If any of the predicates (of the rules 1-5) is true, $prefix_v$ is not a locally greater prefix. Indeed, if a neighbor u (of v) is suspicious and $prefix_u$ is a proper prefix of $prefix_v$, then a neighbor of u has a greater prefix than $prefix_v$, or is changing its $prefix$ according to rule 1 above. By deleting the last bits of $prefix_v$, node v is matching $prefix_v$ to an unknown but greater $prefix$. In all 4 other cases, $prefix_u$ is clearly a greater prefix than $prefix_v$, therefore $prefix_v$ modifies (a limited amount of) its last bits to more closely match $prefix_u$.

Deciding the Output. Additional local computations in lines 12-17 conclude a diffusion phase. Once a candidate's $prefix$ variable is well-formed (i.e., once $id_v = \alpha^{-1}(prefix_v)$), this node becomes a leader. If in later rounds it becomes a follower, then it withdraws from the leader role. Although this process violates the safety property, it is necessary in order to elect a leader, as the last remaining candidate cannot detect that it is the last, due to the lack of termination detection in this preliminary eventual LE version.

Analysis. The 5 rules described above are an idea adopted from [25]. Thus the described information diffusion process satisfies Lemma 30 and Theorem 31 below, adopted from the results of [25] and adapted here to our beeping algorithm (see Section 3.1.4 for more details).

Lemma 30 (Beeping version of Lemma 8 in [25])**.** *Let u and v be two neighboring nodes. Then, $prefix_u$ and $prefix_v$ are identical, except in at most 6 (least significant) bits: without loss of generality, from the $|prefix_u|^{th}$ bit (possibly included) to the $|prefix_v|^{th}$ bit.*
Note that if the $|prefix_u|^{th}$ bit differs in $prefix_u$ and $prefix_v$, then $||prefix_u| - |prefix_v|| < 6$

Theorem 31 (Beeping version of Theorem 10 in [25])**.** *Let X be the maximum identifier. After $|\alpha(X)|+6r$ phases of the information diffusion algorithm, all nodes within distance r (for any $r \geq 0$) from the node with ID X have $prefix = \alpha(X)$. Thus, after at most $|\alpha(X)| + 6D$ phases, for each node v, $prefix_v = \alpha(X)$, and there is a unique candidate node.*

Proof. Let l be the maximum ID node. We prove the theorem by induction on r. Node l has the maximum identifier X, thus it appends a bit from $\alpha(X)$ in each diffusion phase. After $|\alpha(X)|$ phases, $prefix_l = \alpha(X)$. This concludes the case when $r = 0$.
For the induction step ($r > 0$), consider any given node u at distance $r + 1$ of node l, and one of its neighbors v at distance r from l. By Lemma 30, $prefix_u$ and $prefix_v$ differ in less than 6 bits. After $|\alpha(X)| + 6r$ phases, since $prefix_v = \alpha(X)$ (induction hypothesis), node v does not modify $prefix_v$ and node u necessarily corrects (removes, changes or adds) at least one of $prefix_u$'s bits in each of the 6 following phases, until $prefix_u = \alpha(X)$. □

Recall that a communication phase is composed of $c = O(1)$ rounds (c is defined in Section 5.2.3). This implies the following theorem.

Theorem 32. *Uniform Eventual Leader Election is solved by Algorithm 9 in $O(D+\log n)$ rounds (in $\mathcal{BEEP}_S$).*

Proof. Let v be any given node and X the maximum identifier in the network. From Theorem 31, $prefix_v = \alpha(X)$ after $O(D + \log n)$ phases. Nodes have the leader's identifier by applying the α^{-1} function. Moreover, the maximum ID node is well-formed after $|\alpha(X)| = O(\log n)$ phases, thus after $O(\log n)$ rounds. As a result, the maximum ID node is, and remains, a leader henceforth. □

5.2.2 Balanced messages

A basic design technique called multi-slot design pattern [23], allows to communicate constant-size messages without the sender's ID nor multiplicity, given a synchronous start. It works in communication phases of M rounds, if at most M different messages (in $\{1, \ldots, M\}$) are allowed. Beeping in the j^{th} round of a phase is equivalent to sending the message j. However, receivers cannot detect which (and how many) nodes sent that message. Thus, due to the beeping model's restrictions, if a node

sends a message m, it receives no information about whether any of its neighbors also did.

Clearly, this design technique cannot be used to directly send $prefix$ values, as these values are in $\{1, \ldots, N\}$, and communication phases would be $O(N)$ rounds long. But this technique can be adapted to send the values of a locally constrained (k-balanced) variable. A variable var is said to be k-balanced if it satisfies the *k-balancing property*, that is, if the difference between neighboring var values is at most k (for every node v and neighboring node u, $|var_u - var_v| \leq k$).

If one wishes to communicate k-balanced messages, then it is enough to convey, for a message m, the *remainder* $r = m \ mod(1 + 2k)$, using the previous technique, with phases of $M = 1+2k$ rounds (where k is known a priori to all nodes). Then, the receiver knowing at the same time its own remainder, the sender's remainder and the fact that the messages are k-balanced, can deduce the originally sent message (but does not know if multiple nodes have sent this message). Specifically, let v be the receiver and u the sender. Node v deduces the original message m_u from the received remainder message r_u: $m_u = m_v + r_u - r_v - \lfloor \frac{r_u - r_v}{k+1} \rfloor M$.

Received remainder $r_u =$	'0'	'1'	'2'	'3'	'4'	'5'	'6'	'7'	'8'
$m_u - m_v =$ $(r_u - r_v) - \lfloor \frac{r_u - r_v}{k+1} \rfloor M$	-1	v	+1	+2	+3	+4	-4	-3	-2
Decoded message $m_u =$	18	19	20	21	22	23	15	16	17

Figure 5.1 – Communication of k-balanced messages, where $k = 4$ and $M = 9$. The executing node v, and its message value m_v, are highlighted. If v receives a message $r_u = 3$, it is able to deduce that the corresponding message m_u is 21.

Consider the example depicted in Figure 5.1 for $k = 4$. For a given node v, any message m_u sent by a neighboring node u is in $\{m_v - k, \ldots, m_v + k\}$. By conveying the remainder $r_u = m_u \ mod(1+2k)$, node u indicates whether its message m_u is in the next 4 values or in the previous 4, respectively to m_v, and the exact position amongst the 4 possibilities (more precisely, through $r_u - r_v$). The remaining $-\lfloor \frac{r_u - r_v}{k+1} \rfloor M$ factor deals with the fact that some lower (than m_v) messages m_u result in a high remainder r_u, and some higher messages m_u in a low remainder r_u, due to the modulo operation. Node v can deduce the message m_u by using all of this information, along with its own message m_v.

The k-balanced message technique is of independent interest, and allows efficient algorithm design when nodes communicate locally-similar values.

5.2.3 Designing constant-size communication phases

In this section, we show that by applying the balanced messages approach, using only $O(1)$ beeping rounds, a node can deduce its neighbors' $prefix$ values (and whether some of them are suspicious), even though there are $O(N)$ different possible values of $prefix$.

From Lemma 30, we know that $|prefix|$ is a 6-balanced variable. Moreover, two neighboring nodes have similar $prefix$ values, which differ only in (at most 6 of) the last bits. Therefore, if a node can learn the last 6 bits of a neighboring $prefix$, and their exact positions, then it can fill up the empty bits (in more significant positions) using the bits from its own $prefix$. To learn that, one could use two consecutive communication subphases: the first communicates the *position* of the last bit (which is $|prefix|$, a 6-balanced variable) in a subphase with 13 rounds, and the second communicates an *ending* message with the last 6 bits (using a message from $\{1, \ldots, 2^6\}$, encoding all possible 6 letters combinations), in a subphase with 64 rounds. However, this does not work in $\mathcal{BEEP}_S$ because one needs to know, for every different ending message sent by neighbors, the corresponding position of the last bit (thus the corresponding position message). Although this is trivial in $\mathcal{CONGEST}$, because messages from different neighbors are received on different edge ports, it is too costly to simulate this functionality in $\mathcal{BEEP}_S$ (see Section 5.1.1). Fortunately, as the message space is constant-size in both of these communication subphases, the Cartesian product of both message spaces is also constant-size. This allows to associate position and ending messages, using $O(1)$ rounds, even in $\mathcal{BEEP}_S$. Consequently, communication phases with 832 rounds (for messages in $\{1, .., 13\} \times \{1, \ldots, 2^6\}$) are needed to communicate enough information for a node to deduce all neighboring $prefix$ values.

On top of that, the nodes also need to communicate the boolean *suspicious*. For this reason, the message space is adapted to $\{1, .., 13\} \times \{1, \ldots, 2^6\} \times \{\texttt{false}, \texttt{true}\}$. This results in communication phases (introduced in Algorithm 9, Section 5.2.1) of length $c = 1664$ rounds, which although large, is still $O(1)$ size.

5.2.4 Remarks on the eventual leader election algorithm

As mentioned in the related work (Section 5.1.1), [25] is particularly relevant to our work. In this section, we discuss this in detail.
The structure of the information diffusion algorithm is essentially the same. The algorithm progresses in diffusion phases, consisting of a communication phase (corresponding to a single round in the considered $\mathcal{CONGEST}$ model) where nodes send their $(prefix, suspicious)$ values, after which nodes change their $prefix$ variable depending on the $(prefix, suspicious)$ pairs received. Recall the 5 rules presented in Section 5.2.1: the set of the different possible changes for the $prefix$ variable is of a constant size, and these changes are meant to affect at most a constant number of (the last) bits of $prefix$. An important point in [25] is the proof that this set of changes allows the maximum identifier to spread over the network, in an optimal $O(D + \log n)$ number of phases. We use the same constant-size set of changes (for $prefix$). That is why Lemma 30 also applies to our algorithm.

However, the other core element of their information diffusion algorithm, the communication phase, cannot be used in $\mathcal{BEEP}_S$. In [25], nodes maintain up-to-date copies of the $prefix$ variables of their neighbors to circumvent the limited message size and can keep these copies up-to-date in a single communication phase of $O(1)$ rounds. In such a phase, nodes communicate what change was carried out (and which neighbor sent which message): sending the type of change is equivalent

to sending the complete $prefix$ value in this situation. In $\mathcal{BEEP}_S$, nodes are unable to know which neighbor sent which message. Although this capability can be simulated, it is unlikely that it can be done without increasing the time complexity of [25]. Current methods result in an $O(\Delta^4)$ multiplicative factor (see discussion in Section 5.1.1).

One of the main contributions of this work is the introduction of the *k-balanced message* method to leverage the local constraints between (unbounded) values, which allows to communicate in $O(1)$ rounds. With the *k-balanced message* technique, a node can convey a value of $prefix$ to its neighbors in $O(1)$ rounds (of $\mathcal{BEEP}_S$) only. This communication process differs greatly from that of [25].

5.3 Uniform Terminating Leader Election (Explicit LE)

Being often used as a primitive, the LE algorithm must be uniform and detect termination (e.g., so that it can be composed with other algorithms). Since classical approaches are not suited to $\mathcal{BEEP}_S$, we propose an explicit leader election algorithm using a different termination detection approach. Notice that, as mentioned previously, it is simple (in a synchronous setting) to transform the uniform eventual leader election algorithm, Algorithm 9, into a *non-uniform* one using knowledge of D and N, and thus of the time complexity expressed in terms of these parameters. Then, candidates can wait until the algorithm terminates, by counting rounds corresponding to the evaluated time complexity. However, this technique cannot be used here.

Instead, we use a primitive called *overlay networks*. We briefly describe it in Section 5.3.1. Then, in Section 5.3.2, an adapted version of this primitive is used to create a uniform termination detection component. This component is combined with the previously presented eventual leader election algorithm to obtain uniform explicit leader election.

5.3.1 Overlay network

The overlay network approach, in the context of leader election, was first used for $\mathcal{BEEP}_S$ in [58]. Such an overlay has a designated root, and consists of layers centered around the root. Nodes at a distance d from that root (*level* d), have *up links* (resp. *down links*) towards all neighboring nodes (of the overlay) at distance $d-1$ (resp. at distance $d+1$) from the root. Using these (virtual) links, the root can gather information about the network and disseminate it. The default behavior for non-root overlay nodes is to relay any beep received over an up (resp. down) link in some phase p, to all down (resp. up) links in phase $p+1$.

Overlay Phases. In more detail, overlays work in the following way. Time is divided into *overlay phases* of 9 rounds, where each phase consists of 3 subphases of 3 rounds each. The first 3 rounds are called *control* rounds, the next 3 - *up* rounds and the last 3 - *down* rounds. Each set of 3 rounds is numbered from 0 to 2.

When nodes join the overlay, they initialize a *depth* variable (in $\{0, 1, 2\}$), through which they know some information about their layer (and thus how to

communicate with the other layers). The root node *joins* the overlay at a given time, and assigns itself $depth := 0$. The other nodes willing to join the overlay listen in all control rounds. Since overlay nodes beep in the control round *depth* (in all overlay phases), the joining nodes assign themselves $depth = beepHeard+1 (mod 3)$, where $beepHeard$ is the smallest control round in which a beep was heard.

Satisfying Local Constraints. It is important that the *depth* variable satisfies some local constraints, to be guaranteed by the joining process. More specifically, for any distance d and for any given (overlay) node v in level d, all neighboring (overlay) nodes u in level $d-1$ (resp. in level $d+1$) must have $depth_u = depth_v - 1 \ (mod\ 3)$ (resp. $depth_u = depth_v + 1 \ (mod\ 3)$), where $-1 \ (mod\ 3) = 2$.

With this property, nodes can listen over an up link (resp. down link) by listening in up (resp. down) round $depth - 1 \ (mod\ 3)$ (resp. $depth + 1 \ (mod\ 3)$). Moreover, nodes beep over an up link (resp. down link) by beeping in up (resp. down) round $depth \ (mod\ 3)$. In other words, communication through up and down links is the same as sending, or listening for, a *depth* message (using the multi-slot design pattern from [23], described in Section 5.2.2) using the corresponding subphase (a message from $M_{depth} = \{0, 1, 2\}$).

5.3.2 Termination detection component for explicit leader election

We describe the proposed *termination detection component* and its interactions with the eventual leader election algorithm (Algorithm 9). The termination detection component is meant to gather information from the whole network, on whether there are any candidates with a greater α-ID. If there are none, the last candidate terminates and becomes leader. The combined final algorithm structure is given in Algorithm 10.

As in Algorithm 9, time is divided into diffusion phases, but these phases now include an additional *termination detection phase*. A termination detection phase consists of a border detection phase followed by an adapted overlay phase. The *border detection phase* is a communication phase for messages in $M_{prefix} = \{1, .., 13\} \times \{1, \ldots, 2^6\}$, where nodes can detect if any of their neighbors has a different $prefix$ value (similar to the communication in Section 5.2.3). If that is the case for an overlay node (even the root) which has been part of the overlay for more than 6 phases, this node becomes a *border node* (i.e., there exists a neighbor with a different $prefix$ value). The *adapted overlay phase* is a communication phase with 3 subphases, each for messages in $M_{depth} \times M_{prefix}$. Each adapted overlay network is associated to a specific $prefix$ (i.e., that of the overlay's root, necessarily a candidate node). This prefix is used (communicated) so that nodes can detect whether the other endpoint of a down link or up link, is part of the same overlay (i.e., has the same $prefix$). Consequently, different overlay networks do not interfere with each other. A border detection phase has $|M_{prefix}|$ rounds and an adapted overlay phase has $9|M_{prefix}|$ rounds, thus a termination detection phase has $s = 10|M_{prefix}|$ rounds.

Algorithm 10 Uniform Terminating Leader Election Algorithm

```
1: IN: id: identifier ; OUT: leader: boolean, leaderId: identifier
2: candidate := true, prefix := ϵ, suspicious := false      ▷ ϵ is the empty word
3: leaderId := 0, leader := false
4: for diffusion phase p := 1 ; p++ do
5:     // First, a communication phase with c = O(1) rounds.
6:     Communicate (prefix, suspicious) to all neighboring nodes.
7:     // Then, apply predicates of rules 1 to 5 on received
8:     // (prefix, suspicious) pairs.
9:     Use received (prefix, suspicious) values to update prefix, candidate and
10:    suspicious.
11:    // Finally, termination detection phase with s = O(1) rounds.
12:    Execute a termination detection phase.
13:    if candidate and prefix = α(id) then
14:        If no beep is heard in down links, exit the loop.
15:    else
16:        If a beep is heard in up links, exit the loop.
17: leaderId := α^{-1}(prefix)
18: if candidate then
19:    leader := true                          ▷ Last candidate becomes the leader
```

High-Level Description. Upon having a well-formed $prefix$, each candidate designates itself as root and starts constructing an overlay network by using the termination detection phase. Nodes which have just joined the overlay and border nodes beep in their up links (relayed all the way back to the root) using the adapted overlay phase. As a result, the root hears beeps in its down links in each (termination detection) phase, until the overlay network covers the whole graph (Lemma 34). Moreover, the only overlay that can cover the whole graph is the overlay of the highest α-ID node (because this node never changes $prefix$ depending on another node's α-ID, and consequently never joins another candidate's overlay). Therefore, when the root hears no beeps in its down links (and is not a border node), it knows that its overlay covers the whole graph, and that it is the highest α-ID node (thus the maximum ID node). All other roots hear beeps in down links (or become border nodes), until their $prefix$ is changed.

Detailed Overlay Construction. In more detail, the *construction* of the adapted overlay networks is done as follows. Once a candidate node has a well-formed $prefix$ (after exactly $|\alpha(id)|$ diffusion phases), it sets itself up as an overlay's root (in phase $p = |\alpha(id)|$), but it stays silent for 6 termination detection phases (from phase p to phase $p + 5$) before beeping in the control rounds of phase $p + 6$ (and only in this phase). On the other hand, follower nodes with a well-formed $prefix$ attempt to join the overlay corresponding to $prefix$ right away. Once a follower node joins an overlay (in phase p'), it also stays silent for 6 termination detection phases before beeping in the control rounds of phase $p' + 6$.

For any given node v that joins an overlay in termination detection phase p',

its neighbors know if they join v's overlay or not, by phase $p' + 6$ at the latest (by Theorem 31). By staying silent for 6 termination detection phases upon joining, v ensures that all of its neighbors join the overlay at the same time (if they choose to join). Consequently, two nodes u and v, at the same distance d from an overlay's root r, never join r's overlay in different termination detection phases, and $depth_u = depth_v$. Otherwise, we could have $depth_u \neq depth_v$, which means a common neighbor of u and v at distance $d-1$ from r would not have properly defined down links.

Lemma 33. *Let r be the root of an overlay network. This overlay is properly constructed, that is, nodes at level d have the same depth value.*

Proof. Let us prove by induction that if a node at distance d from r joins r's overlay, then it is in phase $|\alpha(id_r)| + 6d$. Let us first consider a node v at distance 1 from r. For node v to join r's overlay, another overlay node must beep in the control rounds and $prefix_v$ must be equal to $\alpha(id_r)$, in the same phase. Notice that for any given two neighbors u and v, which are in different overlays, both nodes beep in different control rounds, because $prefix_u \neq prefix_v$.
In phase $|\alpha(id_r)| + 6$, r beeps in the control rounds, and thus v can join in that phase (if $prefix_v = \alpha(id_r)$). In addition, if $prefix_v \neq \alpha(id_r)$ in phase $|\alpha(id_r)| + 6$, then by Theorem 31, node v does not consider $\alpha(id_r)$ as the highest $prefix$ value it has encountered. As a result, it is impossible that $prefix_v = \alpha(id_r)$ after phase $|\alpha(id_r)| + 6$, and that v joins r's overlay after phase $|\alpha(id_r)| + 6$. The induction step ($d > 1$) is similar, starting from a node v at distance d from r. □

Overlay Property. Because the adapted overlay networks are properly constructed, we can prove that as long as an overlay has not covered the whole network, follower nodes beep in their up links, stopping the root from becoming a leader. In more detail, after a candidate node beeps in the control rounds, it listens to its down links in every termination detection phase. As long as it hears a beep in these links, or is a border node, it does not become leader. Once no beep is heard, it becomes leader, sends a beep in its down links and terminates. On the other hand, after a follower node joins the overlay (in phase p), its beeps in its up links in the first 7 termination detection phases (from phase p to phase $p + 6$). It also beeps in the up links if it is a border node (and relays any beep heard through a down link). Finally, when a follower node hears a beep in its up links, it terminates. Consequently, before an overlay network covers the whole network, the root receives beeps in every (termination detection) phase.

Lemma 34. *Let r be the root of an overlay network. Then from diffusion phase $|\alpha(id_r)| + 6$ onwards, node r hears beeps in its down links every phase, until it becomes a border node itself, or until its overlay covers the whole network.*

Proof. Let r be the root of an overlay network. From Lem. 33, r's overlay network is properly constructed, therefore the virtual links can be used. We define a (overlay) *downwards path* from node v to node u, as a sequence of down links starting in v and ending in u. A node u is *downwards reachable* from node v if there is an overlay downwards path from v to u.

Consider a follower node v, having just joined r's overlay (in phase p). Node v beeps in its up links for 7 termination detection phases after it joins (from phase p to phase $p+6$). For each additional level in the overlay with nodes downwards reachable from v, v beeps in its up links during 7 additional termination detection phases (by relaying beeps heard in its down links, to its up links). Although the next layer (node u) is one further hop away from the root, and starts beeping in phase $p+6$, v beeps during phase $p+6$ (7^{th} termination detection phase after it joins) and relays u's first beep in phase $p+7$. Consequently, there is no interruption in beeps sent through the up links. If an overlay node becomes a border node (some of its neighbors do not join in phase $p+6$), then it beeps in up links in all phases $p' > p+6$. If it exits the overlay, then its neighbors closer to the root become border nodes and beep in their up links. Therefore, the root keeps hearing beeps in its down links while levels are added to its overlay, but also if one of its overlay nodes becomes a border node. In that latter case, the root does not have the maximum ID, and hears beeps in its down links until it becomes a border node itself. □

Theorem 35. *Explicit Leader Election is solved (uniformly) in $O(D+\log n)$ rounds in $\mathcal{BEEP}_S$.*

Proof. The maximum identifier node v starts to construct its overlay network in phase $|\alpha(id_v)|+6$, which is $O(\log n)$. For any given node $u \neq v$, $prefix_v$ never modifies its bits to match $prefix_u$. Consequently, v never joins u's overlay and v's overlay is the only one to grow until it covers the whole network, at a rate of adding a level every 6 diffusion phases. Thus, v's overlay covers the whole network after an additional $O(D)$ diffusion phases. Node v hears beeps in its down links for another additional $O(D)$ phases, since beeps from the last nodes to join the overlay take $O(D)$ rounds to reach v. After that, node v no longer hears beeps in its down links (Lemma 34) and is the only node in the network to terminate as leader. Then, it beeps in its down links, so that all nodes can terminate. □

5.3.3 Discussion and Perspectives

The deterministic LE algorithm presented in Sections 5.2 and 5.3 works without any change with an arbitrary (unbounded) ID space $\{1,\ldots,U\}$. In this case, its round complexity is $O(D+\log U)$. For an unbounded ID space, a known result from distributed bit complexity [47] gives a lower bound of $\Omega(\log U)$ for a network with two nodes. This implies a lower bound of $\Omega(D+\log U)$ for multi-hop networks. Consequently, the presented algorithm is asymptotically time-optimal even with an unbounded ID space.

Furthermore, the algorithm can be modified to work when starting with only a subset of nodes as candidates, or when the IDs are not unique, as long as the highest ID is still unique[2]. Since a set of (non-unique) IDs with a unique maximum can be generated without knowing n or N [81], this last variant can then be applied to obtain a randomized uniform (in both n and D) eventual leader election algorithm.

[2]With non-unique ids, only eventual leader election can be guaranteed.

5.4 Additional Results

LE is an important and often-used primitive when designing distributed algorithms. Thus, it makes sense that improving the time complexity of LE results in improved time complexities for other tasks. We propose improved algorithms for leader election in anonymous networks and MIS and coloring (in trees).

5.4.1 Randomized Leader Election

When dealing with communication-restrictive models such as $\mathcal{BEEP}_S$, anonymity is especially important from an application viewpoint. Indeed, when considering large scale dynamic wireless networks, it might not be economically feasible to equip all nodes with unique identifiers. Additionally, nodes might be prevented from revealing their unique IDs (explicitly or through their actions), due to privacy or security concerns [97]. For this case, a deterministic algorithm assuming unique identifiers can be adapted into a randomized one (w.h.p. time and safety guarantees) for anonymous networks, as stated in [60]. Indeed, one can generate a unique ID w.h.p. by independently sampling $\theta(\log n)$ bits. But in return the knowledge of the network size n or at least some polynomial upper bound $N = O(n^c)$, is required.

However, it is also possible to obtain a (slighly less efficient) randomized uniform (in both n and D) solution. Indeed, the deterministic leader election algorithm still works correctly if nodes' identifiers are from a set of non-unique IDs with a unique maximum ID. Using α-IDs instead of IDs does not affect this property. It is shown in [81] that a set of (non-unique) IDs with a unique maximum can be generated w.h.p. without knowing n or N. The lengths of these randomly generated IDs are in expectation $O(\log n \log^* n)$, and w.h.p. $O(\log n(\log^* n)^2)$, where $\log^* n$ is the iterated logarithm[3]. Therefore, the proposed deterministic leader election solution combined with [81] results in a randomized uniform (in both n and D) $O(D+\log n(\log^* n)^2)$ eventual leader election algorithm (in which both success and time complexity are guaranteed w.h.p.).
For a slightly different result, in which error probability is upper bounded by $\frac{1}{n}$ rather than $\frac{1}{n^{\theta(1)}}$, [81] shows a set of (non-unique) IDs with a unique maximum can be generated (with probability $1-\frac{1}{n}$) in which the lengths of the randomly generated IDs are $O(\log n)$ with probability $1-\frac{1}{n}$. As a result, a randomized uniform (in n and D) $O(D+\log n)$ eventual leader election algorithm can be obtained, in which both success and time complexity are guaranteed with probability $1-\frac{1}{n}$.

5.4.2 MIS and 5-Coloring for Trees

It is well-known that given a leader in tree networks (elected using $O(D+\log n)$ rounds), it is simple to 2-color the tree in $O(D)$ supplementary rounds. However, MIS and coloring have an $\Omega(\log n)$ lower bound (even in tree networks, as the bound from [94] holds for a graph of disconnected pair of nodes), so this $O(D+\log n)$ 2-coloring algorithm is non optimal for most communication graphs. Still, using

[3]The iterated logarithm is defined recursively as: $\log^* n = 1+\log^* \log n$ if $n > 1$, and 0 otherwise.

the proposed uniform leader election algorithm, we design uniform, asymptotically time-optimal $O(\log n)$ *MIS* and *5-coloring* algorithms in $\mathcal{BEEP}_S$, for tree networks.

We first give the algorithmic description of the 5-coloring algorithm. Roughly, low degree nodes are colored first using 3 colors, and the remaining nodes form a subgraph where the connected components have at most a logarithmic diameter. Using the LE algorithm, these connected components can be 2-colored in a logarithmic number of rounds. Now, we give more details as to how these steps are achieved. First, the *LimitedDegreeColoring* algorithm presented in Section 4.4 is used to 3-color all nodes v with $deg(v) \leq 2$, in $O(\log n)$ rounds. Then, since all remaining nodes have a degree of at least 3, every remaining (non-colored) connected component (a tree) has a diameter of at most $\log n$. Thus, electing a leader for each such connected component requires $O(\log n)$ rounds. It is well-known that, in trees, coloring nodes according to their distance to the root can be done using 2 colors. This distance can be learnt by all nodes in $O(\log n)$ rounds. Specifically, nodes are synchronized after the leader election, and the leader broadcasts a beep, using a beep wave [60, 40] or reusing the overlay network from the leader election. The phase in which a node receives the broadcasted beep indicates its distance to the leader. Therefore the remaining non-colored nodes can be colored with another 2 colors, resulting in a 5-coloring for the communication graph.

From this 5-coloring, it is simple to compute an MIS in 5 additional 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. Consequently, an MIS on the communication graph can also be computed in $O(\log n)$ rounds.

Notice that since all parts of the uniform 5-coloring algorithm are themselves uniform, it is a bit tricky to force nodes to resynchronize during the sequential execution. For this purpose, we use the EBET technique presented in Section 4.7, to provide synchronization points in a uniform fashion - that is possible because, for every component of the proposed algorithm, the terminal state at a node can be detected locally - and thus solve the issue.

5.5 Summary

In a first part, we proposed a deterministic and completely uniform (in n and D) leader election algorithm with an $O(D + \log n)$ asymptotically optimal round complexity. Classical approaches used to solve leader election in $\mathcal{CONGEST}$ models do not directly apply to $\mathcal{BEEP}_S$. Although they can be adapted using a transformer, doing so is too costly in most communication graph topologies (see discussion in the related work section: Section 5.1.1). To solve the strongest version of LE, the explicit leader election (defined in Section 2.3), we proceeded in two main steps.

- First, in Section 5.2, we design a uniform algorithm for eventual leader election by building upon the k-balanced message technique.
- Then, in Section 5.3, we combine this algorithm with a specially designed uniform termination detection component to obtain a uniform explicit leader election algorithm.

In a second part, we showed how the proposed deterministic LE algorithm can be applied to obtain additional results in Section 5.4.

- First, in Section 5.4.1, by independently sampling $\theta(\log n)$ bits to create unique identifiers w.h.p. and using the deterministic LE algorithm, we also obtain a uniform (in D only) randomized leader election algorithm which takes $O(D + \log n)$ rounds w.h.p. and works in anonymous networks. However, this solution is not uniform in n. By combining the deterministic LE algorithm with results from [81], a randomized uniform (in both n and D) $O(D + \log n(\log^* n)^2)$ w.h.p. LE algorithm is obtained, which is nearly time optimal.

- Then, in Section 5.4.2, using the deterministic LE algorithm we propose the first asymptotically time-optimal (in $O(\log n)$ rounds) Maximal Independent Set (MIS) and 5-coloring algorithms for trees in $\mathcal{BEEP}_S$ (leveraging the fact that given a leader in a tree network, it is simple to compute a 2-coloring). The MIS and coloring algorithms can be considered as essential symmetry-breaking procedures, and designing optimal-time solutions (even limited to tree networks) might be crucial for other applications in $\mathcal{BEEP}_S$.

Interestingly, both deterministic and randomized (uniform in D) LE solutions presented here are the first to achieve time-optimality for these guarantees in both $\mathcal{BEEP}_S$ and the radio network model with collision detection, outperforming all previous deterministic and randomized results. This work closes the gap between upper and lower bounds for LE.

Chapter 6

Information Dissemination and Data Aggregation

In this chapter, we consider a distributed global communication problem (still in the synchronous starts setting) - *multi-broadcast* - building on top of the global interference control method presented in the previous chapter (i.e., leader election).

In multi-broadcast, k sources (generally with $k << n$) should communicate their messages to the whole network. Variants of the problem include broadcast (with a single source) and gossiping (where all nodes are sources). Without any coordination at a global scale, it is impossible for the k sources to efficiently control interference and communicate their information. To deal with this, leader election can be used to provide a needed degree of global interference control, and allow for the required coordination. A natural solution would now use the leader to coordinate the k sources to broadcast their messages in a strict order, to avoid collisions. However, this results in a non-time-optimal solution.

Instead, we utilize group testing techniques to present time-optimal (and nearly-optimal but more computationally-efficient) solutions. Using such techniques allows sources to broadcast their messages simultaneously, in a coded manner, while limiting the negative impact of collisions (and the resulting information loss). By doing so, the k source messages can be transmitted faster than by relying on k consecutive broadcasts.

6.1 Introduction

We consider a fundamental distributed communication problem: multi-broadcast. Optimal and nearly optimal uniform solutions are presented. Contrary to previous results, these solutions are *constructible*. It is important to emphasize that these results come from an entirely original approach based on *(combinatorial) group testing theory*. Group testing is a method coming from statistics, initially introduced during the Second World War to quickly detect an infection among a group of people [48]. In its original formulation (i.e., as *probabilistic group testing*), the defects were assumed to follow some probability distribution, and the goal was to design a strategy identifying all defects using a small expected number of tests. Probabilistic group testing has been used for local neighbor discovery tasks in some distributed settings [80]. In the *combinatorial* context [78, 71], no assumptions are made about the distribution of the defects and the goal is to design a strategy with a small maximum number of tests (i.e., a worst-case scenario). Results from combinatorial group testing are crucial to the current work. They are used to efficiently detect all

broadcasting sources, since these can be arbitrary, i.e., cannot be assumed to follow some known probability distribution.

Related Work for Multi-Broadcast. In [40], an $O(D \cdot \log L + k \log \frac{LM}{k})$ round deterministic, completely uniform (in L, D and k) algorithm for k-source multi-broadcast is presented, and the lower bound of $\Omega(D + k \log \frac{LM}{k})$ rounds is given. The multi-broadcast algorithm of [40] also provides an $O(n \log \frac{LM}{n} + D \cdot \log L)$ round solution for gossiping. The time-optimal leader election algorithm presented in Chapter 5 is crucial in improving the results of [40]. In a first time, by executing k consecutive leader elections interleaved with the solution of [40], a slightly improved multi broadcast solution can be obtained: $O(D \cdot \log L)$ factors are reduced to $O(D \cdot \min\{k, \log L\})$. Then, in [42], the lower bound for multi-broadcast given in [40] is extended to also apply to randomized algorithms and a time-optimal deterministic and uniform solution to multi-broadcast is proposed. However, this solution relies on a non-constructive existence proof of a complex combinatorial structure, meaning that it must be pre-computed for each possible set of network parameters, and provided to the network nodes in advance (see discussion below).

Importantly, by considering the $\mathcal{BEEP}_S$ model, the focus is on how non-destructive interference impacts the multi-broadcast problem. This improves the understanding of this problem even for stronger models. For example, in the related radio network model assuming $O(\log n)$ bit messages and collision detection, the fastest known (non-explicit) algorithms were designed in $\mathcal{BEEP}_S$ [42]. Somewhat counter-intuitively, efficient solutions for the stronger model do not use the $O(\log n)$ bits contained in the messages, but simply rely on collision detection. In comparison, in radio networks assuming $O(\log n)$ bit messages without collision detection (in which solutions cannot leverage the non-destructive interference for communication), the best multi-broadcasting randomized algorithm [10] requires $O(n \log n)$ time while the best deterministic algorithm [33] requires $O(n \log^4 n)$ time.

Explicitness. Algorithms in $\mathcal{BEEP}_S$ (and related models such as the radio network models) generally seek to minimize the number of rounds required to complete communication tasks. As a result, the cost of local computations is often ignored. Indeed, the fastest deterministic communication algorithms in $\mathcal{BEEP}_S$ and in the radio network models, are often *non-explicit*: they rely upon the use of combinatorial objects whose existence is only proven existentially (see e.g. [41, 42]). Although the existence proofs of the combinatorial objects involved are 'non-constructive', they do imply a naive construction: one can simply generate candidate objects randomly, if shared randomness is available, and in lexicographical order otherwise, and test if they actually satisfy the conditions of the object. However, there are exponentially many possible candidates, and testing naively whether these candidates objects are the required combinatorial objects necessitates an exponential number of computations. Such an approach thus results in an impractically high computation cost.

In some settings an argument can be made that an exponential computation cost may still be acceptable, since the construction of suitable combinatorial objects only *ever* needs to be performed once, and henceforth the object can be stored and provided whenever needed to wireless devices. However, in $\mathcal{BEEP}_S$ this approach

poses a problem: the combinatorial objects that we need depend on the parameters of the network which are not known in advance. Hence, network nodes would have to be pre-loaded with objects for every possible set of parameters. This is again impractical, especially since our aim is to model networks of weak devices which would generally have very limited space.

Consequently, we are only concerned by computationally tractable solutions. In $\mathcal{BEEP}_S$, *explicit solutions* correspond to algorithms with computation time polynomial in L and k (for the nodes), and *weakly explicit solutions* to algorithms with computation time polynomial in L and exponential in k. The latter can still be computationally feasible if $k << L$ when performing multi-broadcast, and thus of practical interest.

6.2 Group Testing

We draw from group testing theory to design efficient solutions in $\mathcal{BEEP}_S$ (see Section 6.4). The objective of *group testing* is to identify a subset of defective items in a set, by testing multiple items at a time instead of resorting to individual testing. One example is the christmas tree lighting problem: to search for a broken bulb among a group of six, one can arrange electrically in series three bulbs and apply a voltage. If they light up, then they are in good condition, and the broken bulb is one of the three others. Some classical applications of group testing are blood testing, DNA library screening, signal processing, streaming algorithms and wireless multiple-access communications [49].

Formal Definition. A formal definition of the (d, I)-*combinatorial group testing* (CGT) problem follows. Consider I items, represented by the integers in $\{1, \ldots, I\}$, and any arbitrary subset B of d items. The items in B are said to be *defective*. The only way to differentiate defective items from good (i.e., non-defective) items is through testing. For efficiency reasons, tests consider sets of items (*pools*) instead of individual items. When testing a pool, a positive result (output 1) indicates that at least one item in the pool is defective, whereas a negative result (output 0) indicates that no item in the pool is defective. Tests are considered to be error-free. A solution to the CGT problem is a *group testing strategy*, that is, a sequence of t tests (for some positive integer t) such that the set B can be computed from the results by using a *decoder*. One way of computing B is to use the *naive decoder*: a set B' is initialized to the set of all items (i.e., $\{1, \ldots, I\}$) after which for every negative test result (output 0), the items of the test's pool are removed from B'. It is important to note that the group testing strategy is tightly related to the decoder: more complex decoders could lead to fewer tests.

Explicitness in Group Testing. In group testing literature, testing strategies are devised to identify defective items from a pool, and efforts have been made to minimize the number of tests, and stages of adaptivity, required by the strategies. Again, however, it transpires that the best deterministic strategies rely on

existentially-proven combinatorial objects, and so are not efficiently constructible or decodable, by the tester.

Consequently, computationally tractable solutions are sought, for practical reasons. In the group testing literature, an *explicit* strategy is one in which each test sequence can be constructed and the output decoded, in time polynomial in I and d. Also of interest is a weaker notion, which we refer to as *weak explicitness*, where construction and decoding time is polynomial in I and exponential in d. The terminology used here corresponds to that used for multi-broadcast. More precisely, when an explicit (respectively weakly explicit) testing strategy is used to obtain a solution to multi-broadcast, the result is an explicit (resp. weakly explicit) algorithm.

Related Work for Group Testing. In the most frequent setting in group testing, *non-adaptive* (i.e., offline) group testing, all tests are designed offline: a test's outcome does not influence the following tests. Non-adaptive group testing allows tests to be performed in parallel. However, it was proven in [53] that test strategies in non-adaptive group testing require $\Omega(d^2 \cdot \frac{\log I}{\log d})$ tests. An explicit construction with $O(d^2 \cdot \log I)$ tests for the non-adaptive setting is given in [91]. On the other hand, in a *fully adaptive setting* (i.e., online setting), where each test's pool depends on the results of all previous tests, the information theoretic lower bound implies that test strategies require $\Omega(d \log \frac{I}{d})$ tests, but all tests must be performed sequentially. An optimal fully-adaptive test strategy is given in [71]. Intermediately, *adaptive* group testing refers to multiple *stages* of tests: all tests of a stage are defined independently from the results of the stage, but can depend on the results of previous stages' tests. Thus tests in the same stage can be done in parallel but successive stages must be treated sequentially. Surprisingly enough when compared with non-adaptive group testing, it is possible to construct two-stage test strategies with $\Theta(d \log \frac{I}{d})$ tests [18, 35]. In particular, a weakly explicit construction for such two-stage testing strategies (with $O(d \log \frac{I}{d})$ tests) is given in [35]. Additionally, explicit constructions are given in [29, 72, 88] with a nearly optimal number of tests. In particular, [88] gives an explicit construction for strategies with $O(d^{1+\epsilon} \log I)$ tests for any value $\epsilon > 0$.

Matrix Notations. For any $a \times b$ matrix M and any integers $i \in \{1, \ldots, a\}$ and $j \in \{1, \ldots, b\}$, the entry of M in row i and column j is denoted by $M[i, j]$. Additionally, the i^{th} row of m is denoted by $M[i, :]$ and the j^{th} column of m is denoted by $M[:, j]$. For any integer d, let I_d be the $d \times d$ identity matrix, that is, the matrix with entry 1 on the diagonal and 0 otherwise.

6.3 A General Scheme for Multi-Broadcast

A natural solution for multi-broadcast is as follows. First, a leader node (with the maximum ID) is elected, allowing the network to rely on broadcast and convergecast (respectively, sending a message from and to the leader). Once a leader has been elected, the ID range L is known to all nodes. Relying on communications via

the leader, it is now possible to efficiently compute global bounds on the network's diameter D and the message range M. Then, the k sources are identified and ordered, as efficiently as possible, by all nodes. Henceforth, this is referred to as the *source identification component*. Finally, the sources convergecast their messages to the leader (pipelined so that the messages arrive to the leader contiguously in order), and the leader broadcasts the string of messages back through the network. Since all nodes agree on the sources' order, all nodes now have all the messages together with the corresponding IDs of the sources. We outline this scheme in Algorithm 11.

Algorithm 11 Multi-Broadcast Scheme

1: Perform Leader Election
2: Estimate Network Parameters
3: Perform Source Identification
4: Collect Source Messages
5: Broadcast Source Messages

All the steps of Algorithm 11, with the exception of Source Identification, can be performed efficiently, explicitly, and deterministically using known procedures from previous works on $\mathcal{BEEP}_S$:

- Leader election can be performed with $O(D + \log L)$ round complexity (see Chapter 5). The algorithm requires unique identifiers and elects the node with the maximum identifier. The output is a boolean indicating whether the executing node is the leader or not.

- Estimating diameter D can be performed in $O(D)$ rounds [42]. The algorithm requires a leader, and outputs in all nodes an estimate $\tilde{D}$ with $D \leq \widetilde{D} \leq 2D$. Henceforth, we assume that D is known because $\widetilde{D}$ can be used instead of D with only a constant-factor overhead.

- Message range M can be similarly estimated in $O(D + \log M)$ time [42].

- Collecting source messages can be done using the CollectMessages procedure from [42]. This procedure requires a leader and upper bounds of D and the maximum length, in bits, of the messages to be collected, denoted by p. It takes as input a set of messages held by nodes in the network. On completion, the leader receives the OR superposition of all the messages, and the running time is $O(D + p)$ rounds.

  We apply this procedure by collecting messages of $p = k\lceil \log M \rceil$ bits, one from each source, in which source numbered i in lexicographical order places its input message into the bit interval $[i\lceil \log M \rceil, (i+1)\lceil \log M \rceil)$, with 0's in every other position (the values of k and the order i are computed during the previously performed source identification component). The superposition of these words is therefore simply the concatenation of all source messages in order. The running time is $O(D + p) = O(D + k \log M)$.

- Broadcasting source messages can be performed using the BEEP-WAVE procedure of [42]. This procedure allows a leader to broadcast a p-bit message to all nodes in $O(D+p)$ time. Applying the procedure to the concatenation of all k source messages in order yields an $O(D+k\log M)$ time.

All these auxiliary procedures terminate such that nodes start each subsequent procedure synchronously. Consequently, source identification is the only remaining step for which there is no efficient procedure, and it is here that the perspective of group testing allows us to make improvements. We denote the round complexity of a potential source identification algorithm by T_{SI}. Efficient deterministic source identification solutions are presented in Section 6.4 and their round complexities are given by Theorems 44 and 47. Moreover, the scheme for source identification when k is unknown is presented in Section 6.4.3. Finally, an efficient randomized source identification solution is presented in Section 6.5 and its round complexity is given by Theorem 49.

Theorem 36. *Multi-broadcast can be solved in $O(D+\log L+k\log M+T_{SI})$ rounds in $\mathcal{BEEP}_S$.*

Proof. Applying the above procedures to the scheme in Algorithm 11, the total running time of steps 1 and 2 is $O(D+\log L+\log M)$. After these steps, a leader is elected and all nodes know common constant-factor upper bounds for D, L and M. The subsequent procedure for source identification takes T_{SI} rounds, and results in all nodes being aware of all source IDs. Finally, steps 4 and 5 are then correctly performed, completing multi-broadcast in a further $O(D+k\log M)$ rounds. The total running time is therefore $O(D+\log L+k\log M+T_{SI})$. □

6.4 Source Identification and Group Testing

We now show how the problem of source identification can be reduced to that of combinatorial group testing (defined in Section 6.2). Recall that we have k source nodes with unique IDs from $[L]$, a specified leader node which is known to all nodes in the network, and universal knowledge of (linear upper bounds on) L and D. Upon completing source identification, we require that the leader node has knowledge of all the source IDs (i.e., of S).

Efficient and simple group testing strategies can be obtained by using *list disjunct matrices* (LDM). Such strategies, called LDM-strategies, are presented in Section 6.4.1 and are the building blocks of the source identification algorithm, described in two stages. First, a simplified scheme (when the number of sources k is known) is presented in Section 6.4.2. Then an extended scheme for unknown k is presented in Section 6.4.3. This extended scheme computes a CLDM-strategy (an extension of an LDM-strategy), and its time complexity (resp. computation cost) depends on the CLDM-strategy's parameters (resp., explicitness property). Weakly-explicit and explicit constructions of CLDM-strategies with optimal or nearly optimal parameters are proposed in Section 6.4.4, resulting in efficiently constructible source identification and multi-broadcast solutions.

6.4.1 Group Testing Strategies and LDM-strategies

Recall that the (d, I)-combinatorial group testing problem (CGT) consists of finding a subset B of d defective items within a set of I items. Good strategies for CGT use at least 2 stages (see Related work in Section 6.2). In a two-stage strategy, a first stage determines a subset B_1 of $\{1, \ldots, I\}$ with $B_1 \supset B$ and $|B_1| = \hat{I}$, and the second stage determines a subset B_2 of $\{1, \ldots, \hat{I}\}$ with $B_2 \supset f_1(B)$ and $|B_2| = d$ (where f_1 maps B_1 to $\{1, \ldots, \hat{I}\}$ in lexicographical order).

Definition 4. *Let B be some unknown subset of d defective items within a set of $\hat{I}$ items. A testing strategy using s stages and t tests over all s stages to determine a superset $B' \supset B$ of size at most $d + \ell - 1$ is called a $(d, \ell, \hat{I})$ s-stage t-test testing strategy.*

In group testing, it is common to build testing strategies by using list disjunct matrices. A single list disjunct matrix defines a single stage testing strategy, and a sequence of s list disjunct matrices defines an s-stage testing strategy (for some integer s).

Definition 5. *A $(d, \ell, \hat{I}, t)$-list disjunct matrix is a $t \times \hat{I}$ binary matrix M such that for any disjoint subsets $T, R \subseteq \{1, \ldots, \hat{I}\}$ with $|T| = d$, $|R| = \ell$, there is a row i of the matrix with $\sum_{j \in T} M[i, j] = 0$ and $\sum_{j \in R} M[i, j] > 0$.*

Lemma 37. *A* $(d, \ell, \hat{I}, t)$-list disjunct matrix *defines a $(d, \ell, \hat{I})$ single stage t-test testing strategy: each row $M[i, :]$ defines the pool of the i^{th} test (for $1 \leq i \leq t$).*

Proof. Let M be a $(d, \ell, \hat{I}, t)$-list disjunct matrix. Assume by contradiction that the t-test testing strategy defined by the t rows of M is not a a $(d, \ell, \hat{I})$ single stage t-test testing strategy. Consider the set B' returned by the naive decoder applied on the results of these t tests: B' is initialized at $\{1, \ldots, \hat{I}\}$ and each negative test eliminates all items involved in the tests from B'. Assume by contradiction that $|B'| \geq d + \ell$. Note that the d defective items are never eliminated by the naive decoder and are thus in B'. For the sake of analysis, we can decompose B' into two disjoint subsets, the defective items B and the remaining items R with $|B| = d$ and $|R| \geq \ell$. From the list disjunctness property of M, there is a row i in M such that $\sum_{j \in B} M[i, j] = 0$ and $\sum_{j \in R} M[i, j] > 0$. As a result, the test corresponding to this row is negative and there is a column $j \in R$ such that $M[i, j] = 1$. Consequently, the naive decoder eliminates one of the items in R, hence a contradiction. □

Definition 6. *A* (d, I)-LDM-strategy *using s stages and t tests is a sequence $M_1, \ldots, M_s$ of list disjunct matrices with parameters $(d, \ell_1, I_1, t_1), \ldots, (d, \ell_s, I_s, t_s)$ satisfying:*

- $I_1 = I$,
- $\ell_s = 1$,
- $d + \ell_i - 1 = I_{i+1}$ *for all* $1 \leq i < s$,
- $\sum_{i \leq s} t_i = t$.

Lemma 38. *A (d, I)-LDM-strategy using s stages and t tests is a $(d, 1, I)$ s-stage t-test testing strategy and thus solves (d, I)-CGT.*

Proof. Consider a (d, I)-LDM-strategy $\mathcal{F}$ using s stages and t tests. Then $\mathcal{F}$ is a sequence of s list disjunct matrices $M_1, \ldots, M_s$ with parameters $(d, \ell_1, I_1, t_1), \ldots,$ (d, ℓ_s, I_s, t_s). By Lemma 37, M_1 defines a single stage t_1-test testing strategy. The naive decoder returns a set B_1 such that the set of defective items $B \subset B_1$ and $|B_1| \leq d + \ell_1 - 1 = I_2$. The items of B_1 are mapped to $\{1, \ldots, I_2\}$ according to their lexicographical order (represented by function f_1). Notice that the defective item set B is mapped to $f_1(B)$ (where $|f_1(B)| = |B|$), and that the subsequent stage seeks to determine a superset B_2 of $f_1(B)$ (and not of B). After which, Lemma 37 is similarly applied to $M_2, \ldots, M_{s-1}$, thus defining functions $f_2, \ldots, f_{s-1}$.
Finally, M_s defines the tests of stage s by Lemma 37 and the naive decoder returns $B_s \subset \{1, \ldots, I_s\}$. Since B_s is a superset of $f_{s-1}(\ldots f_1(B))$ and $|B_s| \leq d+\ell_s-1 = d$, $B_s = f_{s-1}(\ldots f_1(B))$. Therefore, as $B = f_1^{-1}(\ldots f_{s-1}^{-1}(B_s))$ then an s-stage t-test strategy defined by $\mathcal{F}$ solves (d, I)-CGT. □

If d is known then a (d, I)-LDM-strategy can be computed (see Section 6.4.4 for some constructions) and this LDM-strategy defines an s-stage t-test testing strategy solving (d, I)-CGT.

6.4.2 Source Identification for known k

In this section, we give a simplified version (Algorithm 12) of the source identification solution, in which we know the number of sources k. This assumption is removed in the extended scheme presented in Section 6.4.3. Algorithm 12 relies on efficient constructions of LDM-strategies (for example, a 2-stage $O(k \log \frac{L}{k})$ weakly explicit LDM-strategy), which are presented later in Section 6.4.4.

Source Identification Scheme (Algorithm 12). The source identification algorithm first computes a (k, L)-LDM-strategy $\mathcal{F}$ using s stages and t tests (which requires knowing k and L), after which sources are identified in s phases. Let $\mathcal{F} = M_1, \ldots, M_s$ where M_u (for $1 \leq u \leq s$) has parameters (k, ℓ_u, L_u, t_u), $L_1 = L$ and $\ell_s = 1$. Details on constructions of good LDM-strategies are deferred to Section 6.4.4. Using a weakly explicit LDM-strategy results in a weakly explicit source identification solution, and an explicit LDM-strategy in an explicit source identification solution.
Nodes start with no knowledge about which nodes could be the sources, and in each phase they obtain more information by implementing a stage of the group testing strategy defined by $\mathcal{F}$ (see Lemma 38). Let f be initialized to the identity function on $\{1, \ldots, L\}$ in the first phase. The function f is updated so that in every phase u, it renames some of the identifiers in $\{1, \ldots, L\}$ to $\{1, \ldots, L_u\}$ (including all source IDs).

The algorithm executes s phases. In each phase u (for $1 \leq u \leq s$), a node v sets $c_u(v)$ to $M_u[:, f(id_v)]$ (i.e., the $f(id_v)^{th}$ column of M_u) if it is a source, and 0^{t_u} otherwise (see lines 5-6). The superposition w of the words c_u is collected by the leader and then broadcast to all network nodes through the use of the auxiliary functions described in Section 6.3 (see lines 7-8). Consequently, nodes compute $S_u = \{x \in \{1, \ldots, L_u\} \mid x \text{ is included in } w\}$ and update f (see lines 11-

13). More precisely, f is updated to $f_u \circ f$, where f_u renames the elements of S_u to $\{1, \ldots, L_{u+1}\}$ according to their lexicographical order: the y^{th} element of S_u is mapped to y. After all s phases are finished, nodes compute $S = f^{-1}(S_s)$ (see line 14).

Implementation of the Testing Strategy. Each phase u for $1 \leq u \leq s$ implements the stage u of the testing strategy. Nodes use the tests of stage u to determine some subset S_u of $\{1, \ldots, L_u\}$ which contains $f(S)$ (where $|f(S)| = |S|$ because no defective item is eliminated by the naive decoder, see Section 6.4.1). Indeed, the leader collects all messages c_u and broadcasts their superposition w to all nodes, which is the superposition of at most k columns of M_u. Each bit $w[i]$ (for $1 \leq i \leq t_u$) can be seen as the test result of test i of stage u in the testing strategy. In the last phase, S_s is a subset of $\{1, \ldots, L_s\}$ with $|S_s| = k + \ell_s - 1 = k$. Therefore, $S_s = f_{s-1} \circ \ldots \circ f_1(S)$.

Algorithm 12 Source Identification Scheme (with known k)

1: **Inputs**: k and upper bounds for L,M and D
2: Compute $M_1, \ldots, M_s$ and their parameters $(k, \ell_1, L_1, t_1), \ldots, (k, \ell_s, L_s, t_s)$
3: $f := id_v$
4: **for** phase $u := 1$; $u \leq s$; $u{+}{+}$ **do**
5: | **if** v is a source node **then** $c_u := M_u[:, f]$
6: | **else** $c_u := 0^{t_u}$
7: | Collect all binary words c_u by OR superposition into w at the leader
8: | Broadcast the superposition w
9: | Get $S_u = \{x \in \{1, \ldots, L_u\} \mid x \text{ is included in } w\}$
10: | **if** $u < s$ **then**
11: | | Let f_u be a function from S_u to $\{1, \ldots, L_{u+1}\}$ in lexicographical order.
12: | | **if** v is a source node **then**
13: | | | $f = f_u(f)$
14: Return $S = f_1^{-1} \circ \ldots \circ f_{s-1}^{-1}(S_s)$ ▷ S is the set of source IDs

Theorem 39. *Assume Algorithm 12 computes a (k, L)-LDM-strategy $\mathcal{F}$ using s stages and t tests. Then it solves source identification in $O(Ds + t)$ rounds in $\mathcal{BEEP}_S$.*

Proof. Algorithm 12 solves source identification since the testing strategy defined by $\mathcal{F}$ correctly identifies all k source nodes.
In phase u ($1 \leq u \leq s$), the leader gather binary words of t_u bits from the nodes in $O(D + t_u)$ rounds. Then the leader broadcasts the superposition in $O(D + t_u)$ rounds. Over all s phases, the round complexity is $O(\sum_{u \leq s}(D + t_u)) = O(Ds + t)$ rounds. □

Therefore, a good source identification solution should use an LDM-strategy with both small s and small t. The related work in Section 6.2 describes such strategies. However, these either require high computation cost (i.e., weak explicitness) or non-optimal (but nearly optimal) s and t [88].

6.4.3 Extending the Source Identification Scheme to Unknown k

An extended scheme (of Algorithm 12), working when k is unknown, is presented below. The scheme computes an s-stage L-CLDM-strategy (see Definition 8) instead of a (k, L)-LDM-strategy, where the former object is a sequence of constructions that produces an $(\hat{k}, L)$-LDM-strategy for any number of defective items $\hat{k} \leq L$, and can thus be computed when k is unknown. Details on constructions of good CLDM-strategies are deferred to Section 6.4.4.

Definition 7. *A $(\hat{d}, \hat{I})$-list disjunct matrix construction is a function $\mathcal{C}$ with input $(\hat{d}, \hat{I})$ and output (M, ℓ, t) where M is a $(\hat{d}, \ell, \hat{I}, t)$-list disjunct matrix.*

Definition 8. *A I-*CLDM-strategy *is a sequence $\mathcal{C}_1, \ldots, \mathcal{C}_s$ of constructions of list disjunct matrices satisfying: $\forall \hat{d} \leq I$, let $\mathcal{C}_1(\hat{d}, I) = (M_1, \ell_1, t_1)$ and for $1 < i \leq s$, $\mathcal{C}_i(\hat{d}, I_i) = (M_i, \ell_i, t_i)$ for $I_i = \hat{d} + \ell_{i-1} - 1$, then $M_1, \ldots, M_s$ is a $(\hat{d}, I)$-LDM-strategy.*

Scheme for Source Identification with Unknown k. The extended scheme first computes an s-stage L-CLDM-strategy $\mathcal{F}_{\mathcal{C}} = \mathcal{C}_1, \ldots, \mathcal{C}_s$. Following which, sources are identified in s phases, and each phase consists of at most $\lceil \log k \rceil$ subphases. Similarly to Algorithm 12, nodes start with no knowledge about which nodes could be the sources, and in each phase u they obtain more information by implementing at most $\lceil \log k \rceil$ consecutive single stage testing strategies on $\{1, \ldots, L_u\}$. Notice that the set of items $\{1, \ldots, L_u\}$ tested upon does not change throughout the different single stage testing strategies (i.e., subphases) of the phase u. Let f be initialized to the identity function on $\{1, \ldots, L\}$ in the first phase. The function f is updated so that in every phase u, it renames some of the identifiers in $\{1, \ldots, L\}$ to $\{1, \ldots, L_u\}$ (including all source IDs).

Subphase Implementation. In sub-phase r of phase u, if $r = 1$ then node v computes $\hat{k}_u^1$, as the smallest power of 2 ($\hat{k}_u^1 = 2^{g_u}$ for some integer g_u) such that $\mathcal{C}_u(\hat{k}_u^1, L_u) = (M_u^1, \ell_u^1, t_u^1)$ satisfies $t_u^1 \geq D$. This prerequisite ensures that the round complexity of phase u in this extended scheme is the same as that in Algorithm 12. For any other subphase $r > 1$, node v computes $\hat{k}_u^r = 2^{r-1}\hat{k}_u^1$.
Following which, a node v first computes $\hat{k}_u^r$ and $\mathcal{C}_u(\hat{k}_u^r, L_u) = (M_u^r, \ell_u^r, t_u^r)$. Then, it sets c_u to $M_u^r[:, f(id_v)]$ (i.e., the $f(id_v)^{th}$ column of M_u^r) if it is a source, and 0^{t_u} otherwise. The superposition w of the words c_u is collected by the leader and then broadcast to all network nodes through the use of the auxiliary functions described in Section 6.3. Then, nodes compute $S_u^r = \{x \in \{1, \ldots, L_u\} \mid x \text{ is included in } w\}$. If $|B_u^r| \geq \hat{k}_u^r + \ell_u^r$, nodes execute subphase $r + 1$ with $\hat{k}_u^{r+1} = 2\hat{k}_u^r$ and still on items $\{1, \ldots, L_u\}$. Otherwise, nodes finish the current phase and if $u < s$ then nodes execute the following phase $u + 1$ with $L_{u+1} = \hat{k}_u^r + \ell_u^r - 1$ (on items $\{1, \ldots, L_{u+1}\}$) and the function f is updated to $f_u \circ f$, where f_u renames the elements of S_u^r to $\{1, \ldots, L_{u+1}\}$ according to their lexicographical order: the y^{th} element of S_u is mapped to y.

The last subphase of a phase implements the only successful single stage testing strategy of the phase. Moreover, if $k_u^r > k$ then the single stage testing strategy

defined by M_u^r is guaranteed to return a subset S_u^r of less than $\hat{k}_u^r + \ell_u^r - 1$ items. Consequently, each phase has at most $\lceil \log k \rceil$ subphases.

This method can be used to solve (k, L)-CGT with unknown k, at the cost of a multiplicative factor $\lceil \log k \rceil$ for both stages and tests in comparison to the corresponding (k, L)-LDM-strategy computed when k is known. Fortunately, when CLDM-strategies are used in our source identification solution, this multiplicative factor does not affect the round complexity (see Lemma 40 and Th. 41).

Lemma 40. *Each phase u of the extended source identification scheme takes $R_u = O(\sum_{r \leq r'} t_u^r)$ rounds for $r' = \max\{1, \lceil \log k \rceil - g_u\}$. Let t_u be defined by $\mathcal{C}_u(k, L_u)$. If $\mathcal{C}_u$ satisfies $t_u^1 = O(D)$ and if $r' > 1$, $\sum_{r \leq r'} t_u^r = O(t_u)$, then it follows that $R_u = O(D + t_u)$.*

Proof. Consider a phase u (for $1 \leq u \leq s$). The phase takes $R_u = O(\sum_{r \leq r'} t_u^r)$ rounds for $r' = \max\{1, \lceil \log k \rceil - g_u\}$, since in each subphase r (for $1 \leq r \leq r'$), $t_u^r \geq D$ and nodes gather binary words of t_u^r bits at the leader in $O(D + t_u^r) = O(t_u^r)$ rounds, which then broadcasts the superposition in $O(D + t_u^r) = O(t_u^r)$ rounds. □

The conditions of Lemma 40 are satisfied by all 3 CLDM-strategies proposed in Section 6.4.4. Consequently, the following theorem holds for each:

Theorem 41. *Assume that the s-stage L-CDM-strategy $\mathcal{F}_C$ used in the scheme satisfies Lemma 40 for each phase u ($1 \leq u \leq s$). The extended scheme solves source identification with unknown k in $O(Ds + t)$ rounds, where t is defined by the (k, L)-LDM-strategy computed by $\mathcal{F}_C$ (with $\hat{k} = k$).*

Proof. A phase in the extended scheme gives the same correctness guarantees as a phase in Algorithm 12. Therefore, correctness of the extended scheme follows from that of Algorithm 12.
By Theorem 39, Algorithm 12 takes $O(Ds + t)$ rounds. Moreover, by Lemma 40 each phase in the extended scheme has the same round complexity as in Algorithm 12 (given some properties on the CLDM-strategy used). Therefore, the extended scheme takes $O(Ds + t)$ rounds. □

6.4.4 Efficiently constructible source identification solutions

Various CLDM-strategies resulting in efficient deterministic source identification solutions are presented in this section. Theorem 39 from Section 6.4.2 emphasizes that both stages and tests should be as low as possible. However strategies with a single stage require a non-optimal $\Omega(d^2 \cdot \frac{\log I}{\log d})$ tests (see Related work in Section 6.2), thus the CLDM-strategies proposed here have at least 2 stages.
Several constructions of list disjunct matrices are presented, with a trade-off between computational cost and optimal parameters (optimal number of tests). First we give a weakly explicit construction with optimal parameters, resulting in a weakly-explicit (2-stage $O(k \log \frac{L}{k})$-tests) CLDM-strategy and thus a weakly explicit round-optimal source identification solution. Following which, we give two explicit constructions with nearly optimal parameters and use them to construct

two different explicit CLDM-strategies. Their combination results in an explicit nearly optimal (optimal for most ranges of D and k) source identification solution.

Lemma 42. *For any integers $\hat{k}, \hat{L}$ with $\hat{L} > \hat{k}$, the $\hat{L} \times \hat{L}$ identity matrix (i.e., the matrix with entry 1 on the diagonal and 0 otherwise) is a $(\hat{k}, 1, \hat{L}, \hat{L})$-list disjunct matrix. Thus, there exists a construction function $\mathcal{C}_{Ind}(\hat{k}, \hat{L}) = (I_{\hat{L}}, 1, \hat{L})$ with computation cost poly$(\hat{k}, \hat{L})$.*

The matrix construction $\mathcal{C}_{Ind}$ defines a testing strategy with individuals tests on all $\hat{L}$ items. Although this strategy is not efficient when $\hat{L} >> \hat{k}$, it is very efficient once $\hat{L} \leq \hat{k} \log \frac{\hat{L}}{\hat{k}}$. The challenging part is therefore to reduce the L (possibly defective) items to $\hat{L} = O(k \log \frac{L}{k})$ items.

Weakly Explicit Construction with Optimal Parameters. We use an optimal weakly-explicit group testing result from [35]:

Theorem 43 ([35])**.** *There exists a construction function $\mathcal{C}_W(\hat{k}, \hat{L}) = (M_W, \hat{k}, O(\hat{k} \log \frac{\hat{L}}{\hat{k}}))$ with computation cost $O(\hat{k}^3 \hat{L}^{2\hat{k}+1} \log \hat{L})$.*

The CLDM-strategy $\mathcal{F}_1 = \mathcal{C}_W, \mathcal{C}_{Ind}$ is a weakly explicit 2-stage $O(k \log \frac{L}{k})$-test CDLM-strategy. As a side note, $\mathcal{F}_1$ defines what is referred to as a *trivial two-stage testing strategy* in group testing (see Related work in Section 6.2): $\mathcal{C}_W$ determines most non-defective items, after which $\mathcal{C}_{Ind}$ can be used to determine the k defective items (among the remaining $O(k)$ items). When $\mathcal{F}_1$ is given to the source identification scheme in Section 6.4.3, the result is a weakly explicit algorithm with optimal round complexity for source identification.

Theorem 44. *The extended source identification scheme using a testing strategy defined by $\mathcal{F}_1$ is a weakly explicit algorithm solving source identification in optimal $O(D + k \log \frac{L}{k})$ rounds. Consequently, combining this result and the multi-broadcast scheme in Section 6.3, the result is a weakly explicit algorithm solving multi-broadcast in optimal $O(D + k \log \frac{LM}{k})$ rounds.*

Proof. Consider $\mathcal{F}_1 = \mathcal{C}_W, \mathcal{C}_{Ind}$ and the extended source identification scheme presented in Section 6.4.3. It is simple to prove that $\mathcal{C}_W$ and $\mathcal{C}_{Ind}$ satisfy the conditions of Lemma 40. Therefore, we can use Theorem 41 to prove that the extended source identification scheme computing $\mathcal{F}_1$ is a weakly explicit algorithm solving source identification in optimal $O(D + k \log \frac{L}{k})$ rounds. □

Explicit Constructions with Near Optimal Parameters. Unfortunately, there are no known explicit constructions for group testing strategies using $O(k \log \frac{L}{k})$ tests and a constant number of stages. As a result, the best known results in group testing [88] do not give optimal multi-broadcast algorithms in $\mathcal{BEEP}_S$. However, by combining two explicit CLDM-strategies, we can design a multi-broadcast algorithm in $\mathcal{BEEP}_S$ optimal for most ranges of D and k. For $D >> k \log L$ we can use an existing explicit construction from [88]:

Theorem 45 ([88])**.** *For any constant $\epsilon > 0$, there exists a construction function $\mathcal{C}_E(\hat{k}, \hat{L}) = (M_E, \hat{k}^{1+\epsilon}, \hat{k}^{1+\epsilon} \log \hat{L})$ with computation cost $poly(\hat{k}, \hat{L})$.*

For $D << k \log L$ we present a new construction:

Theorem 46. *Given integers $\hat{k}, \hat{L}$ with $\hat{L} \geq 2\hat{k}$, let q denote $\lfloor \log_{2\hat{k}} \hat{L} \rfloor$. There exists a construction function $\mathcal{C}_{DIG}(\hat{k}, \hat{L}) = (M_{DIG}, \hat{k}^q, 2\hat{k}q)$ with computation cost $poly(\hat{k}, \hat{L})$.*

Proof. Write each $j \in [\hat{L}]$ in base $2\hat{k}$, i.e., $j = j_0 j_1 j_2 \dots j_q$, and each digit j_i is an integer between 0 and $2\hat{k} - 1$. For each $x \in [q]$, define the $2\hat{k} \times \hat{L}$ matrix M_x by $M_x[i, j] = 1$ iff $j_x = i$. Then, we let M_{DIG} be the $2\hat{k}q \times \hat{L}$ matrix obtained by vertically concatenating all M_x. We will show that M_{DIG} is a $(\hat{k}, 2^q, \hat{L}, 2\hat{k}q)$-list disjunct matrix.

Let T be a subset of $[\hat{L}]$ with $|T| = \hat{k}$. For each $x \in [q]$, $|DIG_x := \{i : \exists j \in T \text{ with } j_x = i\}| \leq \hat{k}$ i.e., at most $\hat{k}$ different values for digit x are held by the $\hat{k}$ elements of T. For any $j' \in [\hat{L}]$ which has $j'_x \notin DIG_x$, we have $M_x[j'_x, j'] = 1$ and $M_x[j'_x, j] = 0$ for all $j \in T$.
So, for any element j' *not* in the set $DIG := DIG_1 \times DIG_2 \times \dots \times DIG_q$, there is a row in M_{DIG} where j' has value 1 and all elements of T have value 0. DIG is therefore the set of remaining possible defectives, and its size is at most $\hat{k}^q$. □

Two explicit CLDM-strategies are presented here:

- The first strategy $\mathcal{F}_2 = \mathcal{C}_E, \mathcal{C}_{Ind}$ is an explicit 2-stage $O(k^{1+\epsilon} \log L)$-test CLDM-strategy. It is, similarly to $\mathcal{F}_1$, a trivial two-stage testing strategy. When the source identification scheme in Section 6.4.3 uses a testing strategy defined by $\mathcal{F}_2$, the result is an explicit algorithm for source identification with optimal round complexity when $D = \Omega(k^{1+\epsilon} \log L)$.

- The second strategy $\mathcal{F}_3$ is a sequence of $O(\log k \log \frac{\log L}{\log k}) + 1$ constructions, where constructions $\mathcal{C}_i = \mathcal{C}_{DIG}$ for $1 \leq i \leq O(\log k \log \frac{\log L}{\log k})$ and the last construction is $\mathcal{C}_{Ind}$. $\mathcal{F}_3$ is an explicit CLDM-strategy using $O(\log k \log \frac{\log L}{\log k}) + 1$ stages and $O(k \log \frac{L}{k})$ tests. When the source identification scheme in Section 6.4.3 uses a testing strategy defined by $\mathcal{F}_3$, the result is an explicit algorithm for source identification with optimal round complexity when $D = O(\frac{k \log \frac{L}{k}}{\log k \log \frac{\log L}{\log k}})$.

By executing these two source identification solutions (one defined by $\mathcal{F}_2$, the other by $\mathcal{F}_3$) in parallel (i.e., one round of the first algorithm, then one of the second, and so on), the following result can be obtained.

Theorem 47. *Source identification can be solved using an explicit algorithm with optimal round complexity when either $D = O(\frac{k \log \frac{L}{k}}{\log k \log \frac{\log L}{\log k}})$ or $D = \Omega(k^{1+\epsilon} \log L)$ (for any constant $\epsilon > 0$). As a result, multi-broadcast can be solved using an explicit algorithm with optimal round complexity for most ranges of k and D.*

Proof. Consider $\mathcal{F}_2 = \mathcal{C}_E, \mathcal{C}_{Ind}$ and the extended source identification scheme presented in Section 6.4.3. It is simple to prove that $\mathcal{C}_E$ and $\mathcal{C}_{Ind}$ satisfy the conditions of Lemma 40. Therefore, we can use Theorem 41 to prove that the extended source identification scheme computing $\mathcal{F}_2$ is an explicit algorithm solving source identification in nearly optimal $O(D + k^{1+\epsilon} \log L)$ rounds (for any constant $\epsilon > 0$). As a result, if $D = \Omega(k^{1+\epsilon} \log L)$ (for any constant $\epsilon > 0$), then the round complexity above is optimal for source identification.

Similarly, we prove that the extended source identification scheme computing $\mathcal{F}_3$ is an explicit algorithm solving source identification in nearly optimal $O(D \log k \log \frac{\log L}{\log k} + k \log \frac{L}{k})$ rounds. When $D = O(\frac{k \log \frac{L}{k}}{\log k \log \frac{\log L}{\log k}})$, the round complexity above is optimal for source identification. □

6.5 Explicit Solutions for Randomized Group Testing

While asymptotically optimal explicit 2-stage randomized group testing strategies exist (e.g., constructing a $(\hat{d}, O(\hat{d}), \hat{I}, O(\hat{d} \log \frac{\hat{I}}{\hat{d}}))$ list-disjunct matrix by setting each entry to 1 independently with probability $\Theta(1/\hat{d})$), these strategies are not directly implementable in our $\mathcal{BEEP}_S$ framework. This is because they rely on *shared randomness*, i.e., the tester must have access to the randomness used to construct the matrix in order to decode it. However, one practical way to achieve this in $\mathcal{BEEP}_S$ is to have the leader node generate the random bits to be used, and broadcast them to the network. This will result in a time cost (in rounds) equivalent to the number of the generated random bits. To minimize this cost and obtain an efficient randomized multi-broadcast algorithm in $\mathcal{BEEP}_S$, we present a new group testing result demonstrating that an optimal testing strategy can be generated using very few random bits:

Theorem 48. *Given $\hat{d}, \hat{I}$ with $\hat{I} \geq 2\hat{d}$, and $O(\log \hat{I}(1 + \frac{\log \log \hat{I}}{\log \hat{d}}))$ independent uniformly random bits, one can construct an explicit 2-stage group testing strategy $\mathcal{F}_P$ such that for any set T of $\hat{d}$ defective items, the strategy recovers T using $O(\hat{d} \log \frac{\hat{I}}{\hat{d}})$ tests and succeeding with high probability $(1 - 1/poly(\hat{I}))$.*

This strategy can be used in the same source identification framework as those in Section 6.4, starting with an estimate $\hat{k}$ such that $\hat{k} \log \frac{L}{\hat{k}} = \Theta(D)$, and successively doubling until the algorithm succeeds. The resulting algorithm solves source identification in $O(D + k \log \frac{L}{k} + \log L \log \log L)$ rounds, with high probability (i.e., with probability $(1 - 1/poly(L))$).

Theorem 49. *Source identification can be solved in $\mathcal{BEEP}_S$ with an explicit randomized algorithm in $O(D + k \log \frac{L}{k} + \log L \log \log L)$ rounds, succeeding with high probability. This round complexity is optimal whenever $k = \Omega(\log \log L)$.*

6.5.1 Proving Theorems 48 and 49

We first show a construction of a testing matrix to be used in our randomized group testing strategy:

Theorem 50. *Given $\hat{d}, \hat{I}$ with $\hat{I} \geq 2\hat{d}$, and $O(\log \hat{I}(1 + \frac{\log\log \hat{I}}{\log \hat{d}}))$ independent uniformly random bits, one can construct an $O(\hat{d} \log \frac{\hat{I}}{\hat{d}}) \times \hat{I}$ matrix such that the matrix eliminates all but $\hat{d}$ non-defectives with high probability (i.e., with only $1/poly(\hat{I})$ probability of failure).*

Proof. First, classic results on hashing [99] (see definitions below and Theorems 51 and 52) are given. Then, we present a matrix construction that uses only $O(\log \hat{I}(1 + \frac{\log\log \hat{I}}{\log \hat{d}}))$ independent uniformly random bits (leveraging the classic results on hashing). Finally, we prove that this matrix eliminates all but $\hat{d}$ non-defectives with high probability (i.e., with only $1/poly(\hat{I})$ probability of failure).

Definition 9. *A family of functions $\mathcal{H}$ mapping $\{1, \ldots, X\}$ to $\{1, \ldots, Y\}$ is ϵ-almost pairwise independent if for every $x_1 \neq x_2 \in \{1, \ldots, X\}$, $y_1, y_2 \in \{1, \ldots, Y\}$, we have*

$$\mathbf{Pr}\left[H(x_1) = y_1 \text{ and } H(x_2) = y_2\right] \leq \frac{1}{Y^2} + \epsilon \ .$$

Here the randomness is over uniformly random choice of H from $\mathcal{H}$.

Definition 10. *For $X, Y, k \in \mathbb{N}$ with $f \leq X$, a family of functions $\mathcal{G}$ mapping $\{1, \ldots, X\}$ to $\{1, \ldots, Y\}$ is f-wise independent if for every distinct $x_1, \ldots, x_f \in \{1, \ldots, X\}$, the values $G(x_1) \ldots, G(x_f)$ are independent and uniformly distributed in $\{1, \ldots, Y\}$, when G is drawn uniformly at random from $\mathcal{G}$.*

Theorem 51. *There exists an explicit ϵ-almost pairwise independent family $\mathcal{H}$ of functions $H : \{1, \ldots, X\} \to \{1, \ldots, Y\}$ such that any $H \in \mathcal{H}$ can be specified using $O(\log\log \hat{X} + \log \hat{Y} + \log \epsilon^{-1})$ bits.*

Theorem 52. *There exists an explicit f-wise independent family $\mathcal{G}$ of functions $G : \{1, \ldots, X\} \to \{1, \ldots, Y\}$ such that any $G \in \mathcal{G}$ can be specified using $O(f \log XY)$ bits.*

(We omit some details here such as requiring the domain and range of the functions to be integer powers of 2, but since we are concerned with asymptotic complexity, this does not affect the results).

The following functions are used to minimize the amount of random bits necessary for our construction. Let c be a sufficiently large constant. We also require that $\hat{I}$ and $\hat{d}$ are sufficiently large, but again this does not affect asymptotic results.

- Let $\mathcal{H}$ be an explicit $\frac{1}{\hat{d}^3}$-almost pairwise independent family of functions $H : \{1, \ldots, \hat{I}\} \to \{1, \ldots, c\hat{d}\}$ described by Theorem 51, with functions specified using $O(\log\log \hat{I} + \log c\hat{d} + \log \hat{d}^3) = O(\log\log \hat{I} + \log \hat{d})$ bits.
- Let G be an explicit $\frac{4 \log \hat{I}}{\log \hat{d}}$-wise independent family of functions $\{1, \ldots, O(\log \frac{\hat{I}}{\hat{d}})\} \to \{1, \ldots, 2^{O(\log \hat{d} + \log\log \hat{I})}\}$ described by Theorem 52, with functions specified by $O(\frac{4 \log \hat{I}}{\log \hat{d}} \log(\log \frac{\hat{I}}{\hat{d}} \cdot 2^{O(\log \hat{d} + \log\log \hat{I})})) = O(\log \hat{I}(1 + \frac{\log\log \hat{I}}{\log \hat{d}})$ bits.

Randomized Matrix Construction of a Testing Matrix. Importantly, we assume that $O(\log \hat{I}(1+\frac{\log\log \hat{I}}{\log \hat{d}}))$ random bits are provided to the algorithm. Using these, select a random function $g \in G$. Then, for $x \in \{1,\ldots,c\log\frac{\hat{I}}{\hat{d}}\}$, let $H_x : \{1,\ldots,\hat{I}\} \to \{1,\ldots,c\hat{d}\}$ be the function from $\mathcal{H}$ specified by the $O(\log\log \hat{I}+\log \hat{d})$-bit string $g(x)$. We then define a $c\hat{d} \times \hat{I}$ matrix M_x by $M_x[i,j] = 1$ iff $H_x(j) = i$.

Finally, let our testing matrix M be the $c^2\hat{d}\log\frac{\hat{I}}{\hat{d}} \times \hat{I}$ matrix obtained by vertically concatenating all M_x.

Proving the Testing Matrix Property. Let T be our arbitrary set of defective items, i.e., a subset of $\{1,\ldots,\hat{I}\}$ with $|T| = \hat{d}$.

For each $x \in \{1,\ldots,c\log\frac{\hat{I}}{\hat{d}}\}$, let S_x be the set of non-defective items which are not eliminated by a matrix M_y with $y < x$ (i.e., S_x is the set of all items $j' \in \{1,\ldots,\hat{I}\} \setminus T$ such that there is no $y < x$ and $i \le t$ with $M_y[i,j'] = 1$ and $M_y[i,j] = 0\ \forall j \in T$). Then $S_{x+1} \setminus S_x$ is the set of all items which *are* eliminated by matrix M_x. Clearly $S_1 = \{1,\ldots,\hat{I}\} \setminus T$. We now wish to show that for any $x \in \{1,\ldots,c\log\frac{\hat{I}}{\hat{d}}\}$, the probability that $|S_{x+1}| > \frac{|S_x|}{2}$ is at most $\frac{9}{c\hat{d}}$.

Fix some $x \in \{1,\ldots,c\log\frac{\hat{I}}{\hat{d}}\}$. We assume that $|S_x| \ge c\hat{d}$, since otherwise we have already eliminated sufficient items. For $j \in S_x$, denote by $\mathbf{1}_j$ the indicator variable that $j \notin S_{x+1}$, i.e., that j is eliminated by matrix M_x. Notice that by symmetry, these $\mathbf{1}_j$ are identically distributed for all $j \in S_x$ (though they are not independent, or even pairwise independent). Denote the expectation of these indicator variables by μ.

We first bound μ from below. For any item j in S_x, the probability that all elements $j' \in T$ have $H_x(j') \ne H_x(j)$ (i.e., have value 0 on row $H_x(j)$) is lower bounded by:

$$\begin{aligned}\mathbf{Pr}\left[\bigcap_{j'\in T}\{H_x(j') \ne H_x(j)\}\right] &\ge 1 - \sum_{j'\in T}\mathbf{Pr}\left[\{H_x(j') = H_x(j)\}\right] \\ &\ge 1 - \hat{d}\cdot(\frac{1}{c\hat{d}} + \frac{1}{\hat{d}^2}) \\ &\ge \frac{c-2}{c}\ ,\end{aligned}$$

where the initial inequality follows from a union bound and the first equality by $\frac{1}{\hat{d}^3}$-almost pairwise independence of H_x. In this event $\mathbf{1}_j = 1$, so $\mu \ge \frac{c-2}{c}$.

Additionally, by linearity of expectation, $\mathbf{E}\left[|S_x \setminus S_{x+1}|\right] = \sum_{j\in S_x}\mathbf{E}\left[\mathbf{1}_j\right] = \mu|S_x|$. We must now show a concentration bound on $|S_x \setminus S_{x+1}|$. To do so, we will need the following lemma:

Lemma 53. *For any $i \ne j \in S_x$, $\mathbf{E}\left[\mathbf{1}_i\mathbf{1}_j\right] \le \frac{\mu}{2\hat{d}} + \mu^2$.*

Proof.

$$\begin{aligned}
\mathbf{E}[\mathbf{1}_i\mathbf{1}_j] &= \mathbf{Pr}[\mathbf{1}_i = \mathbf{1}_j = 1] \\
&= \mathbf{Pr}[\mathbf{1}_i = \mathbf{1}_j = 1 \wedge H_x(i) = H_x(j)] + \mathbf{Pr}[\mathbf{1}_i = \mathbf{1}_j = 1 \wedge H_x(i) \neq H_x(j)] \\
&\leq \mathbf{Pr}[\mathbf{1}_i = 1 \wedge H_x(i) = H_x(j)] + \mathbf{Pr}[\mathbf{1}_i = \mathbf{1}_j = 1 \mid H_x(i) \neq H_x(j)] \\
&\leq \mathbf{Pr}[\mathbf{1}_i = 1]\,\mathbf{Pr}[H_x(i) = H_x(j)] + \mu^2 \\
&= \mu(\frac{1}{c\hat{d}} + \frac{1}{\hat{d}^2}) + \mu^2 \leq \frac{\mu}{(c-1)\hat{d}} + \mu^2 \ .
\end{aligned}$$

□

We use this bound on the correlation of the indicator variables $\mathbf{1}_j$ to bound the variance of their sum:

Lemma 54. $\mathbf{Var}\left[\sum_{j\in S_x} \mathbf{1}_j\right] \leq \frac{2\mu|S_x|^2}{c\hat{d}}$.

Proof.

$$\begin{aligned}
\mathbf{Var}\left[\sum_{j\in S_x} \mathbf{1}_j\right] &= \mathbf{E}\left[\left(\sum_{j\in S_X} \mathbf{1}_j - \mathbf{E}\left[\sum_{j\in S_x} \mathbf{1}_j\right]\right)^2\right] \\
&= \mathbf{E}\left[\left(\sum_{j\in S_x} (\mathbf{1}_j - \mu)\right)^2\right] \\
&= \sum_{i,j\in S_x} \mathbf{E}[(\mathbf{1}_i - \mu)(\mathbf{1}_j - \mu)] \\
&= \sum_{j\in S_x} \mathbf{E}\left[(\mathbf{1}_j - \mu)^2\right] + \sum_{i\neq j\in S_x} \mathbf{E}[(\mathbf{1}_i - \mu)(\mathbf{1}_j - \mu)] \\
&= |S_x|\mu(1-\mu) + \sum_{i\neq j\in S_x} \mathbf{E}\left[\mathbf{1}_i\mathbf{1}_j - \mu(\mathbf{1}_i + \mathbf{1}_j) + \mu^2\right] \\
&\leq \frac{2}{c}|S_x|\mu + \sum_{j_1\neq j_2\in S_x}\left(\frac{\mu}{(c-1)\hat{d}} + \mu^2 - 2\mu^2 + \mu^2\right) \\
&\leq \frac{2\mu|S_x|^2}{c^2\hat{d}} + \frac{\mu|S_x|^2}{(c-1)\hat{d}} \\
&\leq \frac{2\mu|S_x|^2}{c\hat{d}} \ .
\end{aligned}$$

Here the first inequality uses Lemma 53, and the second relies on our assumption that $|S_x| \geq c\hat{d}$. □

Now that we have a bound on the variance of $\sum_{j\in S_x} \mathbf{1}_j$, we simply apply Chebyshev's inequality to obtain

$$\mathbf{Pr}\left[\left|\sum_{j\in S_x} \mathbf{1}_j - \mu|S_x|\right| \geq \epsilon\right] \leq \frac{\mathbf{Var}\left[\sum_{j\in S_x} \mathbf{1}_j\right]}{\epsilon^2} \leq \frac{2\mu|S_x|^2}{c\hat{d}\epsilon^2} \ .$$

Setting $\epsilon = \frac{\mu|S_x|}{2}$ yields:

$$\mathbf{Pr}\left[\sum_{j\in S_x} \mathbf{1}_j \leq \frac{\mu|S_x|}{2}\right] \leq \frac{8}{c\mu\hat{d}} \leq \frac{8}{(c-2)\hat{d}} \leq \frac{9}{c\hat{d}} \ .$$

So, with probability at least $1 - \frac{9}{c\hat{d}}$, we have $|S_{x+1}| \leq |S_x| - \frac{\mu|S_x|}{2} = \frac{\mu|S_x|}{2}$ as required.

We will call any x with $|S_{x+1}| > \frac{\mu|S_x|}{2}$ *bad*. The random strings used to construct each matrix M_x are $\frac{c\log\hat{I}}{\log\hat{d}}$-wise independent, hence so are the events that each x is bad. Therefore,

$$\begin{aligned}
\mathbf{Pr}\left[\text{at least } \frac{4\log\hat{I}}{\log\hat{d}} \text{ values } x \text{ are bad}\right] &\leq \binom{c\log\frac{\hat{I}}{\hat{d}}}{\frac{4\log\hat{I}}{\log\hat{d}}} \left(\frac{9}{c\hat{d}}\right)^{\frac{4\log\hat{I}}{\log\hat{d}}} \\
&\leq \left(\frac{ce\log\frac{\hat{I}}{\hat{d}}}{\frac{4\log\hat{I}}{\log\hat{d}}}\right)^{\frac{4\log\hat{I}}{\log\hat{d}}} \left(\frac{9}{c\hat{d}}\right)^{\frac{4\log\hat{I}}{\log\hat{d}}} \\
&\leq \left(\frac{9e\log\hat{d}}{4\hat{d}}\right)^{\frac{4\log\hat{I}}{\log\hat{d}}} \\
&\leq 2^{-\frac{4\log\hat{I}}{\log\hat{d}}\log\sqrt{\hat{d}}} \\
&= 2^{-2\log\hat{I}} \\
&= \hat{I}^{-2} \ .
\end{aligned}$$

So, with high probability, at most $\frac{4\log\hat{I}}{\log\hat{d}}$ values x are bad, i.e., at least $\frac{c}{2}\log\frac{\hat{I}}{\hat{d}}$ are not. For any x which is not bad, M_x eliminates at least a half of the remaining non-defective items. Then, the number of items which are not eliminated by the concatenated matrix M is at most

$$\left(\frac{1}{2}\right)^{\frac{c}{2}\log\frac{\hat{I}}{\hat{d}}} |S_1| \leq \hat{I}\cdot 2^{-\frac{c}{2}\log\frac{\hat{I}}{\hat{d}}} \leq \hat{I}\cdot 2^{-\log\frac{\hat{I}}{\hat{d}}} \leq \hat{d} \ .$$

That is, at most $\hat{d}$ non-defective items remain. □

Randomized Source Identification. We can now easily describe our two-stage testing strategy:

Proof of Theorem 48. In stage 1, use the construction from Theorem 50 to rule out all but $O(\hat{d})$ non-defective items, using $O(\hat{d}\log\frac{\hat{I}}{\hat{d}})$ tests. In stage 2, test all of the remaining items individually, using $O(\hat{d})$ tests. The probability that both stages succeed is at least $1 - \hat{I}^{-2}$. □

Finally, we describe how to implement this strategy for source identification in $\mathcal{BEEP}_S$.

Proof of Theorem 49. To perform source identification, the leader node generates $O(\log L \log\log L)$ independent uniformly random bits, and broadcasts them to all nodes in $O(D+\log L \log\log L)$ rounds. This is sufficient randomness to perform the group testing strategy of Theorem 48 with any $\hat{d}$ (and $\hat{I} = L$). Then, we initially set $\hat{k}$ such that $\hat{k}\log\frac{L}{\hat{k}} = D$ (or $\hat{k} = 1$ if $D < \log L$). We repeatedly perform the group testing strategy of Theorem 48, doubling $\hat{k}$ until it successfully identifies all sources. By the argument of Theorem 41, this takes only $O(D + k\log\frac{L}{k} + \log L \log\log L)$ total rounds. Furthermore, since we perform at most $\log k$ iterations of the group testing strategy, the probability that they all execute correctly (and therefore our overall probability of success) is at least $1 - \frac{\log k}{L^2} \geq 1 - \frac{1}{L}$ by a union bound. □

6.6 Summary

In a first part, combinatorial group testing theory is introduced in Section 6.2, and a general scheme for multi-broadcast is presented in Section 6.3 which breaks up multi-broadcast into simpler auxiliary tasks: leader election, estimation of network parameters, source identification and dissemination of source messages. Apart from source identification, all other tasks can be achieved with existing methods (either from Chapter 5 or [40, 42]). As for source identification, it can be seen as a group testing problem, albeit with different constraints from those traditionally considered in existing group testing results, due to the distributed nature of multi-broadcast.

In a second part, in Sections 6.4.1 through 6.4.3, group testing strategies based on *list disjunct matrices* (see Definition 5) are shown to give efficient solutions for multi-broadcast. Finally, several constructions of list disjunct matrices are presented in Sections 6.4.4 and 6.5. Some of them are novel and some are from the existing group testing literature. Using these, we obtain several algorithms for the multi-broadcast task:

- An optimal $O(D + k\log\frac{LM}{k})$-time weakly explicit deterministic algorithm.
- An explicit deterministic algorithm optimal for most ranges of k and D.
- An explicit randomized algorithm optimal for $k = \Omega(\log\log L)$.

Chapter 7

2-hop Communication with Uncoordinated Starts

In Chapter 7, we study the harshest variant of the beeping model, $\mathcal{BEEP}_U$, as opposed to Chapters 4 through 6. Importantly, results obtained in the uncoordinated starts setting are more widely applicable. In particular, they are more tolerant to poorly controlled settings, and thus of more practical interest. For instance, the techniques used for this chapter can be useful in the design of dynamic or self-stabilizing algorithms. Additionally, since biological systems often operate in less coordinated and more dynamic settings, this study may prove useful in order to better understand systems in nature.

We are interested in interference control on a local scale, in this harsher setting. For that reason, we consider *desynchronization* problems, and in particular *2-hop desynchronization*.

To achieve a 2-hop desynchronization solution, nodes need to communicate information to other nodes within distance 2. However, all existing techniques providing such communication require some degree of synchronization between neighboring nodes, unavailable in this setting. Therefore, we introduce an original coding technique, suited to the uncoordinated starts setting, which allows nodes to communicate information simultaneously in an uncoordinated manner. With this technique, nodes can communicate beyond their 1-hop neighborhood - i.e., a 2-hop beep, through which a node can beep and be heard by a listening 2-hop neighbor. In other words, we present a *2-hop communication primitive* - allowing nodes to 2-hop beep - with the prerequisite that nodes know (some upper bound on) the maximum degree Δ.

Utilizing 2-hop beeps, an existing desynchronization algorithm [38] can be converted into a 2-hop desynchronization algorithm. By desynchronizing nodes at distance 2 (instead of distance 1), we obtain the required local interference control, strong enough[1] to simulate a message-passing communication between neighboring nodes (in the uncoordinated starts setting).

7.1 Introduction

Additional Definitions Pertaining to $\mathcal{BEEP}_U$. The model is defined in Section 2.1. However, we give several additional definitions here.

[1] A 1-hop desynchronization solution deals with sender-side collisions, but not with receiver-side collisions (see discussion in Section 1.4).

- For any awake node v in a global round r, the *awake 1-hop neighborhood* of v is denoted by $\mathcal{N}^a(v,r) = \{u \in \mathcal{N}(v) |\, u \text{ is awake in } r\}$. Similarly, the *awake (reachable) 2-hop neighborhood* of v is denoted by $\mathcal{N}_2^a(v,r) = \mathcal{N}^a(v,r) \cup \{u \in \mathcal{N}_2(v) |\, u \text{ is awake in } r \text{ and } \exists w \in \mathcal{N}^a(v,r) \cap \mathcal{N}^a(u,r)\}$.

- For any node v, $\mathcal{H}_v$ denotes the *history* of v, defined as an infinite binary vector: $\mathcal{H}_v[r_v] = 1$ if in some local round r_v, v or one of its neighbors beeped, and $\mathcal{H}_v[r_v] = 0$ if v listened in r_v and no neighbors beeped. The factor $\mathcal{H}_v[r_1, r_2]$ (for local rounds $r_1, r_2 \geq 1$ of v) is said to be the *round sequence* $[r_1, r_2]$ of v.

Specific Related Work. In this setting, Cornejo and Kuhn [38] give a probabilistic algorithm solving the *1-hop desynchronization* problem (or equivalently, the interval coloring problem) in $O(\Delta \log n)$ rounds w.h.p.[2] (n is the number of nodes and Δ is the maximum degree of the communication graph). In this problem, every node is required to determine, from some round onwards, an arithmetic sequence of (global) rounds, disjoint from the sequences determined by its neighbors. The period is $T = O(\Delta)$. The solution in [38] builds upon a probabilistic sender-side collision detection subcomponent, where nodes jitter (i.e., delay their beep) by 1 round with probability $\frac{1}{2}$. This eventually allows a node to determine sequences where its neighbors do not beep while it beeps itself. We extend the construction in [38] to the *2-hop desynchronization* problem. Although the solution also builds upon the probabilistic sender-side collision detection subcomponent from [38], the extension relies on 2-hop communication primitives, implementing communication with nodes at distance 2 (on the square of the communication graph). This case is considerably more complicated, since communicating to a non-neighboring node (within distance 2) requires coordinating with a neighboring node relaying the communication. Moreover, nodes must be careful to relay communication up to distance 2 and no further. Although solving 1-hop desynchronization allows nodes to avoid sender-side collisions, receiver-side collisions still remain. Two neighbors of a node v, at distance 2 of each other, can have the same sequence of rounds in a 1-hop desynchronization solution. On the contrary, 2-hop desynchronization requires for the sequence of a node to be disjoint from the sequences of nodes in its 2-hop neighborhood. This allows nodes to avoid both sender and receiver-side collisions. Hence, it is possible to implement higher level communication primitives for sending and receiving messages (cf. concluding remark).

Related Work on Superimposed Codes. In $\mathcal{BEEP}_S$ (with simultaneous wake ups), [60] encodes messages as sequences of beeps and listenings using *superimposed codes* [73] in order to solve leader election efficiently. This technique allows multiple nodes to transmit messages simultaneously, given that they start in the same round. To do so, superimposed codes guarantee that any (OR) superposition (see Section 2.2) of (a limited number of) codewords can be uniquely decoded: the *unique decomposition* property. However, in $\mathcal{BEEP}_U$, nodes may wake up in different (arbitrary)

[2] with probability at least $1 - \frac{1}{n}$.

rounds and have no access to a common source of time. Therefore, nodes cannot ensure that message transmissions start in the same round and superimposed codes cannot be used. In this work, we introduce a variant of this technique that we call *uncoordinated superimposed codes* and which similarly allows simultaneous transmissions (i.e., ensures the unique decomposition property), but in the more general case in which nodes do not start transmitting in the same round. In other words, uncoordinated superimposed codes tolerate arbitrary shifts of (a limited number of) codewords.

Similar combinatorial structures have been used for the wake-up problem in radio networks [59, 34, 32, 31]. In this problem, nodes wake up at some arbitrary times or upon hearing a message. However, upon hearing a collision, nodes do not wake up. Therefore, in an efficient solution, nodes must use messages very carefully in order to avoid collisions and wake up the network's other nodes. For that reason, [34] introduces *radio synchronizers*. These guarantee that for any superposition of (a limited number of) *different* codewords, arbitrarily shifted, there is at least one position at which a single codeword has a 1 bit and all others have a 0 bit. In other words, if nodes communicate according to *unique* codewords with arbitrary shifts, then there is at least one round in which a single node communicates alone and thus avoids collisions.
Importantly, notice that the property provided by radio synchronizers is weaker than that provided by uncoordinated superimposed codes. For instance, radio synchronizers were not designed for superpositions that comprise of the same codeword multiple times, with different shifts. Since dealing with these superpositions is crucial in the proposed solution, uncoordinated superimposed codes are used in this work instead of radio synchronizers.

7.2 Implementing 2-hop Communication Primitives

In this section, the BEEP2H and LISTEN2H primitives are presented. When executed in G, they simulate the effect of BEEP and LISTEN on the square communication graph G^2, albeit with a time delay δ for some positive integer δ. Below we define what it means to implement these primitives.

Definition 11 (BEEP2H and LISTEN2H)**.** *A node v is said to solve (or implement) the BEEP2H and LISTEN2H primitives (with some delay $\delta \geq 0$) if it computes the history of communication on the square communication graph - an infinite binary vector $\mathcal{H}_v^2$ - in the following way:*

- *$\mathcal{H}_v^2$ is initialized to 0 for every element.*
- *If a node v invokes BEEP2H in some round r_v, then $\mathcal{H}_v^2(r_v) := 1$ and $\forall u \in \mathcal{N}_2^a(v, g_v(r_v))$ s.t. u invokes LISTEN2H in r_u (where $r_u = g_u^{-1}(g_v(r_v))$), u sets $\mathcal{H}_v^2(r_u)$ to 1 at the latest in round $r_u + \delta$.*
- *If a node v invokes LISTEN2H in some round r_v, then if $\exists u \in \mathcal{N}_2^a(v, g_v(r_v)))$ s.t. u invokes BEEP2H in $g_u^{-1}(g_v(r_v))$, v sets $\mathcal{H}_v^2(r_v)$ to 1 at the latest in round $r_v + \delta$.*

Encoding Distances as Sequences of Beeps and Listenings. Beeps communicate a very limited amount of information. This obstacle is commonly circumvented by relying on sequences of beeps and listenings (encoded by 0's and 1's) to transmit more complex information, where 0 corresponds to a listening (node executes LISTEN) and 1 corresponds to a beep (BEEP). If a node beeps and listens according to a binary sequence m, the node is said to transmit m (or beep m). These sequences must be carefully chosen, because if several neighbors of a given node transmit such sequences simultaneously, what is received is their superposition. Then the node must be able to extract the different superimposed sequences. This problem can be solved using *superimposed codes* [60] (see Definition 12), but only in the case of simultaneous wake-ups (meaning nodes are somehow *coordinated*). For dealing with the case of arbitrary (or uncoordinated) wake-ups (i.e., $\mathcal{BEEP}_U$), we propose here a variant of this technique that we call *uncoordinated superimposed codes* (see Definition 13 and Figure 7.1).

Definition 12 (from [60])**.** *An (h,k)-superimposed code or SI(h,k) code of length l is a set of h binary codewords of length l such that (1) every superposition of k or less different codewords has a unique decomposition in the set of codewords and (2) every superposition of more than k different codewords is different from any superposition of k or less different codewords.*

Definition 13. *An (h, k)-uncoordinated superimposed code or USI(h, k)-code C of length l is a set of h binary codewords of length l such that (1) every superposition of k or less different offsetted codewords of the form $0^o||c||0^{2(l-1)-o}$ (denoted by (c, o), where $c \in C$ and $o \in \{0, \ldots, 2(l-1)\}$) has a unique decomposition in the set of offsetted codewords, and (2) every superposition of more than k different offsetted codewords is different from any superposition of k or less different offsetted codewords.*

The following lemma plays a crucial role in the design of 2-hop communication primitives (in Section 7.2.2). Basically, the superposition of $k-1$ or less different offsetted (i.e., shifted) codewords (from some USI(h, k)-code C) has a unique decomposition into C, and therefore cannot hide any offsetted codeword other than the superposed $k-1$ offsetted codewords.

Lemma 55. *Consider a USI(h, k)-code C of length l. For any given codeword $c' \in C$, consider the superposition s of $k-1$ or less different offsetted codewords of the form $0^o||c||0^{2(l-1)-o}$ such that $c \in C$, $o \in \{0, \ldots, 2(l-1)\}$, and $o = l-1 \Leftrightarrow c \neq c'$. Then $0^{l-1}||c'||0^{l-1}$ is not included in s.*

Proof. Consider the superposition s' of $k-1$ different offsetted codewords and $(c', l-1)$: s' is a superposition of k different offsetted codewords. By contradiction, assume $0^{l-1}||c'||0^{l-1}$ is included in s. Then s and s' are identical, which contradicts part 1 of Definition 13. □

High-level Description of Algorithm in Section 7.2.2. In this algorithm, 2-hop beeps (beeps over the square communication graph) are transmitted with a

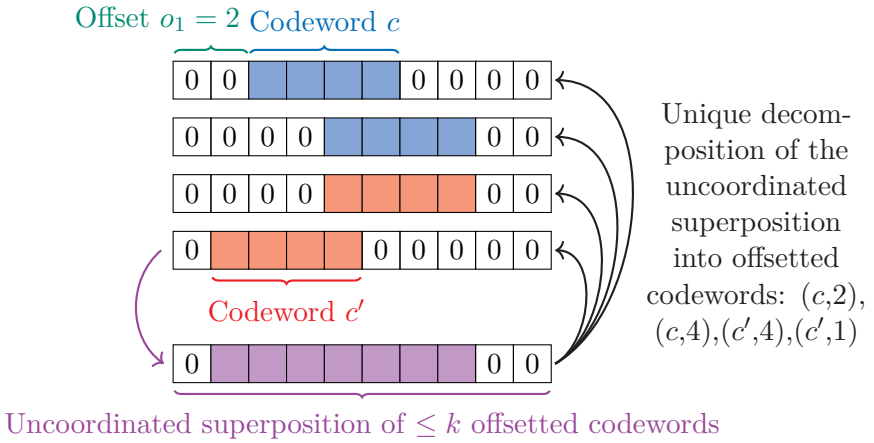


Figure 7.1 – $\{c, c'\}$ is a USI(2,k)-code of length 4. The uncoordinated superpositions of at most 4 offsetted codewords have a unique decomposition into the set of offsetted codewords.

time delay δ for some even integer $\delta \geq 1$. The main idea behind the algorithm is to provide to all nodes the same two codewords c_1 and c_2, which are finite-length sequences (of length l) of zeros and ones. It is important that collisions between uncoordinated transmissions of codewords do not affect the decoding, and for that reason, a USI(2,k)-code (its construction is given in Section 7.2.1) is chosen for $\{c_1, c_2\}$, with some well-chosen integer k. By encoding the distance from any node executing BEEP2H using c_1 (distance 0: source) and c_2 (distance 1: relay), nodes communicate over distance 2 with a delay of $\delta = 4l$ rounds. The *source* s of a 2-hop beep transmits c_1. By the properties of USI-codes, neighbors of s detect the transmission of c_1, which they *relay* by transmitting c_2. When detecting c_1 or c_2, a node learns that it is at most 2 hops away from a 2-hop beep's source.

7.2.1 Using Distinct Difference Sets to construct USI-Codes

In this section, *distinct difference sets* (DDS) are defined and used as building blocks for USI-codes. A construction for distinct difference sets (Theorem 56), derived from Singer's work on finite projective geometry [98], is given in Section 5.5 of [46]. This construction uses prime numbers. By the Bertrand-Chebyshev theorem [28], for every positive integer k, there is a prime number p such that $k \leq p \leq 2k$. As a result, for any integer k, a construction of USI(2,k)-codes of length $O(k^2)$ can be obtained. This construction gives a USI(2,$(\Delta_{up}+1)^2+1$)-code of length $O(\Delta_{up}^4)$ for any given integer value Δ_{up} (Corollary 61), a crucial component for the algorithms of the 2-hop communication primitives presented in Section 7.2.2.

Definition 14 (Distinct Difference Set)**.** *An (l, k) distinct difference set (DDS) is a subset D of $\{0, \ldots, l-1\}$ of size k, such that every non-zero element of $\{0, \ldots, l-1\}$ can be expressed exactly once as $d_1 - d_2 \bmod l$, where $d_1, d_2 \in D$ and $d_1 \neq d_2$.*

Theorem 56 ([98])**.** *Let k be an integer such that $k-1$ is a power of a prime*

number. Then there exists a (constructible) set of k integers $\{d_0, \ldots, d_{k-1}\}$, such that $\forall i \in \{0, \ldots, k-1\}$, $d_i \leq k^2 - k$, and this set is a $(k^2 - k + 1, k)$-DDS.

Given a DDS, a USI-code with a single codeword can be obtained. The codeword leverages the DDS's properties to guarantee the unique decomposition property of the resulting USI-code.

Theorem 57. *Let D be an (l, k)-DDS. Define the set S as the set of integers $\{d + 1, d \in D\}$, and the codeword c as a binary word of length l such that $c[p] = 1$ if $p \in S$. Then $\{c\}$ is a USI(1, $k-1$)-code of length l.*

The following lemma states that if a (l,k)-DDS is used to construct the codeword of a USI(1,k)-code, then any two offsetted codewords (differing only in the offsets) have at most one position where their bits are both 1. As a result, the superposition s of $k-1$ offsetted codewords has at most $k-1$ positions where the bit is 1, in common with a different offsetted codeword a. Therefore, if a is not one of the $k-1$ superposed offsetted codewords, then a is not included in the superposition s, which guarantees the unique decomposition property of the USI(1,k)-code.

Lemma 58. *Let D be an (l, k)-DDS. Define the set S as the set of integers $\{d + 1, d \in D\}$, and the codeword c as a binary word of length l such that $c[p] = 1$ if $p \in S$. For $o, o' \in \{0, \ldots, 2(l-1)\}$, $o \neq o'$, let $a = 0^o||c||0^{2(l-1)-o}$ and $a' = 0^{o'}||c||0^{2(l-1)-o'}$. There exists at most one $j \in \{1, \ldots, 3l-2\}$ such that $a[j] = a'[j] = 1$.*

Proof. By contradiction, assume that there exist $j, j_2 \in \{1, \ldots, 3l-2\}$, $j \neq j_2$ such that $a[j] = a'[j] = 1$ and $a[j_2] = a'[j_2] = 1$. Then, there exist $d_1, d_2, d'_1, d'_2 \in D$, where $d_1 \neq d'_1$, $d_2 \neq d'_2$, $d_2 > d_1$ and $d'_2 > d'_1$, such that $d_2 - d_1 = d'_2 - d'_1$. That contradicts the fact that D is a DDS. □

Proof of Theorem 57. Let us prove by contradiction that every superposition of $k-1$ or less different words $a_i = 0^{o_i}||c||0^{2(l-1)-o_i}$ has a unique decomposition. Consider an arbitrary superposition s of $k-1$ or less different words a_i. Assume that for some $o' \neq o_i$ $\forall i \in \{1, \ldots, k-1\}$, the word $a' = 0^{o'}||c||0^{2(l-1)-o'}$ (which is not part of the offsetted codewords superposed to obtain s) is included in s. By Lemma 58, $\forall i \in \{1, \ldots, k-1\}$, there exists at most one $j \in \{1, \ldots, 3l-2\}$ such that $a_i[j] = a'[j] = 1$. However, $\{p,\ a'[p] = 1\}$ has cardinality k, so a' cannot be included in s.

Therefore, for any superposition s of at most $k-1$ offsetted codewords, by the unique decomposition property, there are at most $k-1$ offsetted codewords included in s. Consequently, the superposition s_2 of $k' \geq k$ different offsetted codewords is different from any superposition of $k-1$ or less different offsetted codewords because at least k' offsetted codewords are included in s_2 (possibly more than k'). □

It follows (Theorem 59 below) that a USI-code with multiple codewords can be obtained by dividing the single codeword from the previous construction, into multiple smaller codewords. The proof of Theorem 59 can be easily obtained from Lemma 60 and the proof of Theorem 57. The proof of Lemma 60 is similar to that of Lemma 58. Corollary 61 results from Theorems 56 and 59.

Theorem 59. *Let D be an $(l, h \cdot k)$-DDS. Define the set S as the set of integers $\{d+1, d \in D\}$, the sets S_1,...,S_h as partitions of S where each S_i is of size k, and for any $i \in \{1, \ldots, h\}$ the codeword c_i as a binary word of length l such that $c_i[p] = 1$ if $p \in S_i$. Then $\{c_1, \ldots, c_h\}$ is a USI(h,k − 1)-code of length l.*

Lemma 60. *Let D be an $(l, h \cdot k)$-DDS. Define the set S as the set of integers $\{d+1, d \in D\}$, the sets S_1,...,S_h as partitions of S where each S_i is of size k, and for any $i \in \{1, \ldots, h\}$ the codeword c_i as a binary word of length l such that $c_i[p] = 1$ if $p \in S_i$. For $o, o' \in \{0, \ldots, 2(l-1)\}$ and $c, c' \in \{c_1, \ldots, c_h\}$, let $a = 0^o||c||0^{2(l-1)-o}$ and $a' = 0^{o'}||c'||0^{2(l-1)-o'}$ such that if $c = c'$ then $o \neq o'$, then there exists at most one $j \in \{0, \ldots, 3l-2\}$ such that $a[j] = a'[j] = 1$.*

Corollary 61. *For any given h and k, one can construct a USI(h,k) of length $l \leq h^2(2k+1)^2$. Thus, for any given k, one can construct a USI(2,k) of length $l \leq 4(2k+1)^2 \leq 16(k+1)^2$.*

7.2.2 Algorithms for the 2-hop Communication Primitives

We consider a communication graph with maximum degree Δ, where an upper bound on the maximum degree $\Delta_{up} = O(\Delta)$ is known by all nodes. Moreover, we assume the knowledge of some integer $f \geq 1$ and of a USI(2,k)-code $C = \{c_1, c_2\}$ of length $l = O(\Delta_{up}^4)$ with $k \geq f(\Delta_{up}+1)^2 + 1$. This code is known to every node of the graph.

Algorithms implementing BEEP2H and LISTEN2H are given below (Algorithms 13 and 14). Both algorithms rely on Algorithm 15, which manages the transmission of codewords c_1,c_2 of the given USI(2,k)-code (using beeps and silences) and the decoding of these codewords (by inspecting the round sequences, defined in Section 7.1).

2-hop Communication Details. The primitives work as follows. A node v invokes either BEEP2H or LISTEN2H in each local round r_v, but cannot invoke BEEP2H more than f times within $4l-1$ rounds.

- On the one hand, v 2-hop beeps in some round r_v by invoking BEEP2H (i.e., 2-hop beeps) in r_v. One says that v starts a 2-hop beep in (global) round $g_v(r_v)$, and v conveys this information to its neighbors by transmitting the codeword c_1 (beeping or listening according to ones or zeros in the binary word c_1).

- On the other hand, v listens to 2-hop beeps (and relays them if necessary) in the following manner. In each local round r_v, if c_1 (resp. c_2) is included in the round sequence $[r_v - 2l, r_v - l - 1]$ of v and $r_v \geq 2l+1$ (resp. $r_v \geq 4l+1$), v sets $\mathcal{H}_v^2[r_v - 2l]$ to 1 (resp. for c_2, sets $\mathcal{H}_v^2[r_v - 4l]$ to 1). Furthermore, v transmits another codeword c_2 (relaying the information that a neighboring node transmitted c_1 previously) to relay the 2-hop beep (resp. transmits nothing).

Constraints on *listStarting*. For some node v, assume v invokes BEEP2H in some local round r_v. Then r_v is stored in $listStarting_v$ so that v knows how it should transmit c_1 in the $l-1$ following rounds. Additionally, the *listStarting* variable is heavily used in the analysis of Algorithm 15 to prove that multiple sequences of BEEP2H and LISTEN2H of different nodes are transmitted correctly. The constraints on the rounds in this list are described in the following remark.

Remark 1. For any given node v and local round $r'_v \in listStarting(v)$: $|\{r''_v \in listStarting(v) \mid r''_v \neq r'_v \text{ and } |r'_v - r''_v| < 4l\}| < f$

Algorithm 13 BEEP2H

1: **IN:** $\{c_1, c_2\}$: USI(2,k)-code of length l, f: maximum frequency, r: current (local) round
2: **INOUT:** *listStarting*: list of integers, $\mathcal{H}^2$: vector of booleans
3: *// Store the round of BEEP2H invocation in memory.*
4: **if** $|\{r' \in listStarting \mid r \neq r' \text{ and } |r - r'| < 4l\}| < f$ **then**
5: | *// At most f BEEP2H invocations allowed within $4l$ rounds for a node.*
6: | $listStarting := listStarting \cup \{r\}$
7:
8: *// Rounds of all BEEP2H invocations within $4l$ rounds transmitted to Alg. 15.*
9: CodewordTransmission($\{c_1, c_2\}$, r, *listStarting*, $\mathcal{H}^2$) ▷ Alg. 15

Algorithm 14 LISTEN2H

1: **IN:** $\{c_1, c_2\}$: USI(2,k)-code of length l, f: maximum frequency, r: current (local) round
2: **INOUT:** *listStarting*: list of integers, $\mathcal{H}^2$: vector of booleans
3: CodewordTransmission($\{c_1, c_2\}$, r, *listStarting*, $\mathcal{H}^2$) ▷ Alg. 15

High-Level Proofs. A high-level analysis of Algorithm 15 is given below, with the proofs deferred to Section 7.2.3. Most importantly, Theorem 68 states that BEEP2H and LISTEN2H (as defined by Algorithms 13 and 14) satisfy Definition 11 given at the beginning of Section 7.2 (with a delay δ of $4l = O(\Delta_{up}^4)$), albeit with a constraint on the frequencies of BEEP2H invocations: a single node cannot invoke BEEP2H more than f times within $4l-1$ rounds when using Algorithms 13 and 14.

Lemma 62. *For any given node v and local round r_v, if v beeps in round r_v then there is a round r'_v, where $r_v - (l-1) \leq r'_v \leq r_v$, such that v transmits c_1 or c_2 starting from r'_v.*

Proof. From the definition of Algorithm 15 (lines 9 and 17). □

In Algorithm 15, the most difficult part consists in listening to, and relaying, 2-hop beeps. This amounts to correctly detecting codeword transmissions by listening to all round sequences $[r, r+l-1]$ of length l. However the round sequence $[r, r+l-1]$ is impacted by any codeword transmission starting in the l-interval

Algorithm 15 CodewordTransmission

```
1: IN: {c_1, c_2}: USI(2,k)-code of length l, r: current (local) round
2: INOUT: listStarting: list of integers, H^2: vector of booleans
3: // With the constraint:
4: // ∀r_1 ∈ listStarting, |{r_2 ∈ listStarting | r_1 ≠ r_2 and |r_1 − r_2| < 4l}| < f
5:
6: // Transmit a bit of codeword c_1.
7: for integer j := 1 ; j ≤ l ; j++ do        ▷ At most f transmissions of c_1 by v.
8:     if (r + 1 − j) ∈ listStarting and c_1[j] = 1 then
9:         BEEP
10:
11: // Listen to beeps to detect bits of c_1.
12: for integer i := 1 ; i ≤ l ; i++ do
13:     r' := r − 2l + 1 − i
14:     // If c_1 detected (starting in r'), v relays information by transmitting c_2.
15:     if c_1 is included in round sequence [r', r' + l − 1] then
16:         if c_2[i] = 1 then
17:             BEEP
18:
19: // If c_1 or c_2 were detected in round sequence [r − 2l, r − l − 1], then update H^2.
20: if c_1 is included in round sequence [r − 2l, r − l − 1] then
21:     H^2[r − 2l] := 1
22: else if c_2 is included in round sequence [r − 2l, r − l − 1] and r ≥ 4l + 1 then
23:     H^2[r − 4l] := 1
```

centered on r (see definition below). Such codewords cannot be avoided, and are unpredictable, since nodes communicate in an uncoordinated manner, making this very challenging. Additionally, a node may be relaying a 2-hop beep and listening for other 2-hop beeps at the same time. In the following, we prove that communication using (codewords from) a USI-code (with the correct parameters) allows nodes to correctly decode transmissions (thus listen to, and relay, 2-hop beeps).

Definition 15. *For any given global round $r \geq 1$, the l-interval centered on r is defined as the set of positive integers in $\{r-(l-1), \ldots, r+(l-1)\}$ and denoted by $I(r)$. Alternatively, for any awake node v and local round $r_v \geq 1$, the notation $I(r_v)$ is used as shorthand for $I(g_v(r_v))$.*

Theorems 63 and 64 provide an upper bound on the number of different codeword-round pairs transmitted during $I(r)$. Importantly, such upper bounds allow to prove Lemmas 65, 66 and 67. More concretely, Theorem 63 states that v transmits at most $f(\Delta+1)$ different codeword-round pairs (i.e., starts a 2-hop beep or relays a 2-hop beep) in $I(r)$ for any round r. As for Theorem 64, it asserts that nodes in the neighborhood of v transmit at most $f(\Delta+1)^2$ different codeword-round pairs in $I(r)$ for any round r. This second theorem is crucial to our results, as it gives an upper bound on the number of uncoordinated codeword transmissions during $I(r)$. Therefore, the properties of our USI(2,k) (with $k \geq f(\Delta+1)^2+1$) can be leveraged

to ensure that nodes properly decode the uncoordinated codeword transmissions.

Theorem 63. *For any given global round $r \geq 1$ and awake node v in r, there are at most $f(\Delta+1)$ different codeword-round pairs (c,r') with $c \in \{c_1, c_2\}$ and $r' \in I(r)$, such that v transmits c starting from round $g_v^{-1}(r')$.*

Theorem 64. *For any given global round $r \geq 1$ and awake node v in r, there are at most $f(\Delta+1)^2$ different codeword-round pairs (c,r') with $c \in \{c_1, c_2\}$ and $r' \in I(r)$, such that a node $u \in \mathcal{N}^a(v, r')$ transmits c starting from round $g_u^{-1}(r')$.*

Lemmas 65, 66 and 67 give a more high-level understanding of Algorithm 15, which is used to prove Theorem 68. Broadly, these lemmas affirm that nodes cannot communicate or receive false codewords in spite of the uncoordinated transmissions (since codewords come from a USI-code with the correct parameters).

- Lemma 65 states that a codeword (i.e., c_1 or c_2) is included in a round sequence of length l if and only if at least one neighboring node transmitted the entire codeword (either started or relayed a 2-hop beep). Consequently, this lemma rules out the possibility that multiple nodes each beep only some part of a false codeword. In other words, multiple nodes cannot create a false codeword through uncoordinated tranmissions.

- Lemma 66 asserts a similar result: for any node v, the codeword c_1 is included in its beep history (starting in some round r) if and only if v starts a 2-hop beep in round r (and thus invoked BEEP2H in round r). That is to say, a node never falsely starts a 2-hop beep.

- Finally, Lemma 67 states that a node v relays a 2-hop beep (by transmitting c_2 starting in global round r) if and only if one of its neighbors u was awake and started a 2-hop beep in global round $r - 2l$. This means that a node never falsely relays a 2-hop beep.

Lemma 65. *For any given global round $r \geq 1$ and awake node v in r: $\forall i \in \{1, \ldots, l\}$ where $c[i] = 1$, $\exists u \in \mathcal{N}^a(v, r)$ s.t. u beeps in round $g_u^{-1}(r) + i - 1 \Leftrightarrow \exists u \in \mathcal{N}^a(v, r)$, $\forall i \in \{1, \ldots, l\}$ where $c[i] = 1$, u beeps in round $g_u^{-1}(r) + i - 1$.*

Lemma 66. *For any given global round $r \geq 1$ and awake node v in r: $\forall i \in \{1, \ldots, l\}$ where $c_1[i] = 1$, v beeps in $g_v^{-1}(r)+i-1 \Leftrightarrow g_v^{-1}(r) \in listStarting(v)$.*

Lemma 67. *For any given global round $r \geq 2l + 1$ and awake node v in r: ($\forall i \in \{1, \ldots, l\}$ where $c_2[i] = 1$, v beeps in $g_v^{-1}(r) + i - 1$) $\Leftrightarrow g_v^{-1}(r) \geq 2l + 1$ and $\exists u \in \mathcal{N}^a(v, r - 2l)$, $g_u^{-1}(r - 2l) \in listStarting(u)$.*

With these results, nodes can deduce the starting rounds of codeword transmissions that started in their awake 2-hop neighborhood, by inspecting round sequences of length l. Consequently, leveraging this property, nodes can communicate over distance 2, albeit with a delay of $4l$ rounds - see Theorem 68.

Theorem 68. *For any given node v, assume that Algorithm 15 is executed for some local round $r_v \geq 4l + 1$. For any local round $r'_v \in \{1, \ldots, r_v - 4l\}$: $\mathcal{H}_v^2(r'_v) = 1 \Leftrightarrow \exists u \in \mathcal{N}_2^a(v, g_v(r'_v))$, s.t. $g_u^{-1}(g_v(r'_v)) \in listStarting(u)$.*

Proof. Assume that $\mathcal{H}_v^2(r'_v) = 1$. Then either c_1 is included in round sequence $[r'_v, r'_v+l-1]$ or c_2 is included in round sequence $[r'_v+2l, r'_v+3l-1]$ (lines 20-21). The first case is equivalent to $\forall i \in \{1,\ldots,l\}$, where $c_1[i] = 1$, $\exists u \in \mathcal{N}^a(v, g_v(r'_v))$ such that u beeps in round $g_u^{-1}(g_v(r'_v))+i-1$. Then, by Lemmas 65 and 66, the first case is equivalent to $\exists u \in \mathcal{N}^a(v, g_v(r'_v))$ such that $g_u^{-1}(g_v(r'_v)) \in listStarting(u)$. The second case is equivalent to $\forall i \in \{1,\ldots,l\}$, where $c_2[i] = 1$, $\exists u \in \mathcal{N}^a(v, g_v(r'_v+2l))$ such that u beeps in round $g_u^{-1}(g_v(r'_v+2l)) + i - 1$. Then, by Lemmas 65 and 67, the second case is equivalent to $\exists u \in \mathcal{N}_2^a(v, g_v(r'_v))$ such that $g_u^{-1}(g_v(r'_v)) \in listStarting(u)$. □

7.2.3 Detailed Proofs of the 2-hop Communication Primitive

The proofs of Theorems 64 and 63, as well as Lemmas 65, 66 and 67, are quite involved. The underlying reason is that the statements are intertwined. We concentrate on proving Theorem 64, using (interleaved) strong induction. The inductive step can be split into two main parts. First, if Theorem 64 holds for all rounds $r_1 \leq r$, then Lemma 67 holds for all rounds $r_2 \leq r + l$. Following which Theorem 64 holds for round $r + 1$, thereby completing the inductive step.

The first step - the conditional proof of Lemma 67 - is the more complex one. We show that if Theorem 64 holds for some round r_1, then Lemmas 65 and 66 also hold for round r_1. Moreover, if Theorem 64 as well as Lemmas 65 and 66 hold for all rounds $r_2 \leq r_1$ (for some round r_1), then Theorem 63 holds for round $r_1 + l$. Finally, if Theorem 64 and Lemma 66 hold for all rounds $r_2 \leq r_1$ (for some round r_1), and Theorem 63 holds for round $r_1 + l$, then Lemma 67 holds for round $r_1 + l$.

Proof: Theorem 64 holds for round $r_1 \geq 1$ ⇒ ***Lemma 65 holds for round*** r_1.
(⇐) Trivial.
(⇒) We transform a problem on concurrent codeword transmissions into an equivalent problem on offsetted codewords, and thus leverage the properties of the USI-code $\{c_1, c_2\}$. Consider the round sequence $I(r_1)$ of v (shorthand for round sequence $[g_v^{-1}(r_1) - (l-1), g_v^{-1}(r_1) + (l-1)]$). It is equal to the first $2l-1$ bits of the superposition s of the offsetted codeword (c',o') defined as follows: for each round $r' \in I(r_1)$ and codeword $c' \in \{c_1, c_2\}$, such that there exists a node $u \in \mathcal{N}^a(v, r')$ and u transmits c' starting from $g_u^{-1}(r')$, a corresponding offsetted codeword (c',o') with $o' = r' - r_1 + l - 1$ is considered (note that $o' \in \{0,\ldots,2(l-1)\}$).
It is known that $\forall i \in \{1,\ldots,l\}$ where $c[i] = 1$, $\exists u \in \mathcal{N}^a(v, r_1)$ s.t. u beeps in round $g_u^{-1}(r_1) + i - 1$. Assume by contradiction that no node $u \in \mathcal{N}^a(v, r_1)$ transmits c starting from $g_u^{-1}(r_1)$. Equivalently, $\forall u \in \mathcal{N}^a(v, r_1)$, $\exists i \in \{1,\ldots,l\}$ where $c[i] = 1$, s.t. u does not beep in $g_u^{-1}(r_1)+i-1$. In terms of offsetted codewords, the offsetted codeword (c,$l-1$), or equivalently $0^{l-1}||c||0^{l-1}$, is included in the superposition s of all offsetted codewords (c',o') considered above (and (c,$l-1$) is not part of the superposition).
Theorem 64 holds for r_1. Thus, there are at most $f(\Delta+1)^2$ different codeword-round pairs (c',r') with $c \in \{c_1, c_2\}$ and $r' \in I(r_1)$ s.t. a node $u \in \mathcal{N}^a(v, r')$ transmits c' starting from $g_u^{-1}(r')$. Consequently, s is the superposition of at most $f(\Delta + 1)^2$

offsetted codewords (c',o') (excluding (c,$l-1$)). Because $\{c_1, c_2\}$ is a USI(2,k)-code with $k \geq f(\Delta_{up}+1)^2+1$, then by Lemma 55, $0^{l-1}||c||0^{l-1}$ is not included in s. Thus $0^{l-1}||c$ is not included in round sequence $I(r_1)$, resulting in a contradiction. □

Proof: Theorem 64 holds for round $r_1 \geq 1$ ⇒ ***Lemma 66 holds for round*** r_1.
(⇐) Follows from the definition of Algorithm 15 (line 9).
(⇒) In a slight departure from the proof above, only transmissions of v (started in $I(r_1)$) are considered. The superposition s of the offsetted codeword (c',o') is defined as follows: for each round $r' \in I(r_1)$ and codeword $c' \in \{c_1, c_2\}$, such that v transmits c' starting from $g_v^{-1}(r')$, a corresponding offsetted codeword (c',o') with $o' = r' - r_1 + l - 1$ is considered (note that $o' \in \{0, \ldots, 2(l-1)\}$).
It is known that $\forall i \in \{1, \ldots, l\}$ where $c_1[i] = 1$, v beeps in round $g_v^{-1}(r_1) + i - 1$. Assume by contradiction that $g_v^{-1}(r_1) \notin listStarting(v)$. Therefore, v does not transmit c_1 starting from round $g_v^{-1}(r_1)$. In terms of offsetted codewords, the offsetted codeword (c,$l-1$), or equivalently $0^{l-1}||c||0^{l-1}$, is included in the superposition s of all offsetted codewords (c',o') considered above (and (c,$l-1$) is not part of the superposition).
Similarly to the proof above, it is shown that since Theorem 64 holds for r_1 and $\{c_1, c_2\}$ is a USI(2,k)-code with $k \geq f(\Delta_{up}+1)^2+1$, we have a contradiction. □

Proof: Theorem 64 holds for all rounds $r_2 \leq r_1$***, with*** $r_1 \geq 1$ ⇒ ***Theorem 63 holds for round*** $r_1 + l$.
Assume by contradiction that there are at least $f(\Delta+1)+1$ different rounds $r' \in I(r_1+l)$, such that v transmits c_2 starting from r'. Therefore, for at least $f(\Delta+1)+1$ different rounds $r' \in I(r_1+l)$, c_1 is included in the round sequence $[g_v^{-1}(r') - 2l, g_v^{-1}(r') - l - 1]$ of v (lines 15-17 of Algorithm 15): v transmits c_2 only if it previously received a c_1 transmission by one of its neighbors.
Lemmas 65 and 66 imply that at least $f(\Delta+1)+1$ local rounds (corresponding to $f(\Delta+1)+1$ global rounds within an interval of $2l-1$ rounds) are part of the *listStarting* variables of neighbors of v, which induces a contradiction with Remark 1. For at least $f(\Delta+1)+1$ different rounds $r'' \in I(r_1 - l)$ ($r'' = r' - 2l$), $\forall i \in \{1, \ldots, l\}$, where $c_1[i] = 1$, $\exists u \in \mathcal{N}^a(v, r'')$ s.t. u beeps in $g_u^{-1}(r'') + i - 1$. Since Theorem 64 holds for all rounds $r_2 \leq r_1$, so do Lemmas 65 and 66. Therefore, for at least $f(\Delta+1)+1$ different rounds $r'' \in I(r_1 - 2l)$, $\exists u \in \mathcal{N}^a(v, r'')$ s.t. $r'' \in listStarting(u)$. Since $|\mathcal{N}(v)| \leq \Delta+1$, $\exists u \in \mathcal{N}(v)$ s.t. there are $f+1$ different global rounds $r_3, \ldots, r_{f+2} \in I(r_1 - 2l)$ with $g_u^{-1}(r_3), \ldots, g_u^{-1}(r_{f+2}) \in listStarting(u)$.
Finally, there are at most $f(\Delta+1)$ different rounds $r' \in I(r_1+l)$, such that v transmits c_2 starting from $g_v^{-1}(r')$. Additionally, by Remark 1, there are at most f different rounds $r'' \in I(r_1+l)$, such that v transmits c_1 starting from $g_v^{-1}(r'')$, and any transmission of c_1 during $I(r_1+l)$ by v is done at the expense of a transmission of c_2 in $I(r_1+l)$. Therefore, there are at most $f(\Delta+1)$ different rounds $r' \in I(r_1+l)$, such that v transmits a codeword starting from $g_v^{-1}(r')$. □

Proof: Theorem 64 holds for all rounds $r_2 \leq r_1$***, with*** $r_1 \geq l+1$ ⇒ ***Lemma 67 holds for round*** $r_1 + l$.
(⇐) It is known that $\exists u \in \mathcal{N}^a(v, r_1 - l)$, $g_u^{-1}(r_1 - l) \in listStarting(u)$. Since

Theorem 64 holds for $r_2 = r_1 - l$, so does Lemma 66. Thus, $\exists u \in \mathcal{N}^a(v, r_1 - l)$,$\forall i \in \{1, \ldots, l\}$ where $c_1[i] = 1$, u beeps in $g_u^{-1}(r_1 - l) + i - 1$. And c_1 is included in the round sequence $[g_v^{-1}(r_1) - l, g_v^{-1}(r_1) - 1]$ of v. By lines 15-17 of Algorithm 15, $\forall i \in \{1, \ldots, l\}$ where $c_2[i] = 1$, v beeps in $g_v^{-1}(r_1 + l) + i - 1$.

($\Rightarrow$) Consider the round sequence $I(r_1 - l)$ of v. It is equal to the first $2l - 1$ bits of the superposition s of the offsetted codeword (c',o') defined as follows: for each round $r' \in I(r_1 - l)$ and codeword $c' \in \{c_1, c_2\}$, such that there exists a node $u \in \mathcal{N}^a(v, r')$ and u transmits c' starting from $g_u^{-1}(r')$, a corresponding offsetted codeword (c',o') with $o' = r' - r_1 + 2l - 1$ is considered (note that $o' \in \{0, \ldots, 2(l - 1)\}$).
Assume by contradiction that $\forall u \in \mathcal{N}^a(v, r_1 - l)$, $g_u^{-1}(r_1 - l) \notin listStarting(u)$. Following which, assume by contradiction that c_1 is included in the round sequence $[r_1 - l, r_1 - 1]$ of v. In terms of offsetted codewords, the offsetted codeword (c_1,$l - 1$), or equivalently $0^{l-1}||c_1||0^{l-1}$, is included in the superposition s of all offsetted codewords (c',o') considered above (and (c_1,$l - 1$) is not part of the superposition). Since Theorem 64 holds for $r_2 = r_1 - l$ and $\{c_1, c_2\}$ is a USI(2,k)-code with $k \geq f(\Delta_{up} + 1)^2 + 1$, we have a contradiction, and c_1 is not included in the round sequence $[r_1 - l, r_1 - 1]$ of v (meaning v does not transmit c_2 starting from round $g_v^{-1}(r_1 + l)$).
Now, it could still be the case that $\forall i \in \{1, \ldots, l\}$ where $c_2[i] = 1$, v beeps in $g_v^{-1}(r_1 + l) + i - 1$. But Theorem 64 holds for r_1, thus Theorem 63 holds for $r_1 + l$. Because $\{c_1, c_2\}$ is a USI(2,k)-code with $k \geq f(\Delta_{up} + 1)^2 + 1$, then by Lemma 55, $0^{l-1}||c_2$ is not included in round sequence $I(r_1 + l)$ of v, resulting in a contradiction. □

Proof of Theorem 64. Theorem 64 is proven trivially for rounds in $\{1, \ldots, l+1\}$ because no node can beep according to c_2 starting from rounds before $2l + 1$ (lines 15-17 of Algorithm 15).

Let r be some round, $r \geq l+1$. For any given node v, assume Theorem 64 holds for all rounds $r_1 \leq r$. Let us prove it also holds for $r + 1$ (inductive step). First, by Remark 1, there are at most $f(\Delta + 1)$ different rounds $r' \in I(r + 1)$, such that $\exists u \in \mathcal{N}^a(v, r')$ and u transmits c_1 starting from $g_u^{-1}(r')$.
Now, assume by contradiction that there are at least $f(\Delta^2 + 1) + 1$ different rounds $r' \in I(r + 1)$, such that $\exists u \in \mathcal{N}^a(v, r')$ and u transmits c_2 starting from $g_u^{-1}(r')$. Since Theorem 64 holds for all $r_1 \leq r$, Lemma 67 holds for all $2l + 1 \leq r_2 \leq r + l$ (and $r' \leq r + l$). By Lemma 67, there are at least $f(\Delta^2 + 1) + 1$ different rounds $r'' \in I(r + 1 - 2l)$ ($r'' = r' - 2l$) such that $\exists w \in \mathcal{N}_2^a(u, r'' - 2l)$ and $r'' \in listStarting(w)$. Thus, $\exists w \in \mathcal{N}_2^a(v, r'')$ s.t. there are $f + 1$ different global rounds $r_3, \ldots, r_{f+2} \in I(r + 1 - 2l)$ and $g_w^{-1}(r_3), \ldots, g_w^{-1}(r_{f+2}) \in listStarting(w)$. By Remark 1, this is not possible. Thus there are at most $f(\Delta^2+1)$ different rounds $r' \in I(r+1)$ such that $\exists u \in \mathcal{N}^a(v, r')$ and u transmits c_2 starting from $g_u^{-1}(r')$. □

7.3 Solving the 2-hop Desynchronization Problem

Now that nodes have access to 2-hop communication primitives, communication on the square graph can be simulated. The solution presented in [38] is extended here

through the use of these primitives, in order to obtain a 2-hop desynchronization. More precisely, in $O(\Delta_{up}^4 \log n)$ rounds after its wake-up, a node determines a periodic sequences of rounds, of period $T = O(\Delta_{up}^4)$, disjoint from those of nodes in its 2-hop neighborhood. Probabilities are necessary to break symmetry between nodes without relying on identifiers.

Now the solution is presented. It is assumed that nodes are given the same period $T = \kappa(\Delta_{up} + 3)^4$ and the same USI(2,$(\Delta_{up} + 1)^2 + 1$)-code C of length $l \leq 16(\Delta_{up} + 3)^4$ ($l = O(\Delta_{up}^4)$), where κ is set to 93. Conceptually, the algorithm is executed in periods of T rounds.
Some supplementary notations are needed. The discrete uniform distribution on a set S is denoted by $\mathcal{U}(S)$. The round number within a period is denoted by index $i \in \{1, \ldots, T\}$. The (local) period numbers are denoted by p and any variable var changes at most once every period, in the round of index $4l$. By abusing the notation the value of var at the start of the round of index $4l + 1$ is said to be the value of var in p and is denoted by $var(p)$. Additionally, for any node v, the starting local round of a period p_v is denoted by $start_v(p_v)$ and the global round related to the local round of index i in period p_v is denoted by $g_v(i, p_v) = g_v(start_v(p_v)) + (i - 1)$.

Algorithm Description. Consider a node v. Upon wake-up, it listens for T rounds (the first period). In any other period $p_v \geq 2$, v listens for the first $4l$ rounds, thus at the end of the round of index $4l$ v has a complete history $B_v(p_v)$ of BEEP2H invocations during its previous period $p_v - 1$. Additionally, v invokes BEEP2H exactly once per period (in $g_v(i_v^b, p_v)$), after computing a round index $i_v^e \in \{4l + 1, \ldots, T - 2\}$, a bit $jitter_v \in \{0, 1\}$ and $i_v^b = i_v^e + jitter_v$. The bit $jitter_v$ is used to obtain probabilistic sender-side collision detection, with probability $\frac{1}{2}$.

Node v computes i_v^e as follows. First, with $B_v(p_v)$, it computes whether any 2-hop neighbors invoked BEEP2H within 2 rounds of $g_v(i_v^b, p_v) - T$. If none did, then $disjoint(p_v) = \texttt{true}$ and v keeps the same round index $i_v^e(p_v) = i_v^e(p_v - 1)$: v starts an arithmetic sequence with common difference T. Else if 2-hop neighbors invoked BEEP2H within 1 round of $g_v(i_v^b, p_v) - T$, then a collision is detected (with probability $\frac{1}{2}$) and $disjoint(p_v) = \texttt{false}$. Following this v randomly computes a round index $i_v^e \in \{4l + 1, \ldots, T - 2\}$, such that the global round $g_v(i_v^e, p_v)$ is at least three rounds away from any global round $g_v(i, p_v)$ for $i \in B_v(p_v)$. As a result, v decides on a sequence disjoint from those of its 2-hop neighbors with a constant probability.

Analysis. The subsequent results are in the same vein as those in [38]. We prove that for any node v, $O(\log n)$ periods after waking up in (global) round r_w, node v is good (see Definition 16) w.h.p. and thus desynchronized w.h.p. with all nodes in its 2-hop awake (reachable) neighborhood $\mathcal{N}_2^a(v, r_w)$. Therefore, $O(\log n)$ periods after all nodes wake up, all nodes output a correct 2-hop desynchronization solution w.h.p. Unfortunately, although the original 1-hop desynchronization algorithm is probabilistically self-stabilizing (i.e., converges to a correct configuration from any given initial configuration, with some probability), the 2-hop desynchronization algorithm presented here is not (due to the use of the proposed USI-codes).

Algorithm 16 2-hop Desynchronization

IN: κ: integer, Δ_{up}: upper bound on the maximum degree, C: USI-code of length l
OUT: *sequence*: sequence of rounds ▷ Sequence determined by v
$disjoint := \texttt{false}$, $lS := \emptyset$, $T = \kappa(\Delta_{up}+3)^4$
// $\mathcal{H}^2$ is the history of communication in the square graph, with a $4l$ delay.
$\mathcal{H}^2 := vector < binary >$
for local round $r := 1$; r++ **do**
  $i := (r-1) \,\%\, T + 1$ ▷ Round index in the current T-round period.

  // Communicate on the square graph
  if $r \leq T$ or $i \leq 4l$ **then**
    LISTEN2H(C,1,r,lS,$\mathcal{H}^2$)
  else if $r > T + 4l$ and $i = i^b$ **then**
    BEEP2H(C,1,r,lS,$\mathcal{H}^2$)
  else
    LISTEN2H(C,1,r,lS,$\mathcal{H}^2$)

  // Local computation once per period $p_v > 1$,
  // at the end of the round of index $4l$.
  if $r > T$ and $i = 4l$ **then**
    // B contains the indexes of BEEP2H invocations during the previous
    // period.
    $B := \{j \in \{1,\ldots,T\} |\, \mathcal{H}^2[r+j-i-T] = 1\}$
    // The free indexes F are indexes at least 3 rounds away from B.
    $F := \{j' \in \{4l+1,\ldots,T-2\} \mid \forall j \in B, |j'-j| > 2\}$
    if $r > 2T$ **then**
      // During previous period, v invoked BEEP2H in rS.
      $rS := r + i^b - i - T$
      if $(i^b - 1) \in B$ or $(i^b + 1) \in B$ **then**
        $disjoint := \texttt{false}$ ▷ If a collision is detected.
      else if $\{rS-2, rS-1, rS+1, rS+2\} \cap B = \emptyset$ **then**
        $disjoint := \texttt{true}$. ▷ If no 2 hop neighbors within 2 rounds.
    if not *disjoint* **then**
      $i^e := \mathcal{U}(F)$ ▷ Choose a new round for the output sequence.
    $jitter := \mathcal{U}(\{0,1\})$ ▷ Compute a jitter bit for sender-side CD.
    // v invokes BEEP2h in round of index i^b of this period.
    $i^b := i^e + jitter$
    $sequence := sequence \cup \{r + i^e - i\}$

Definition 16. *For any given node v and any given period p_v, v is said to be a* good node *in p_v if $\forall u \in \mathcal{N}_2(v)$, for any integer $j \geq 1$, $|sequence_v[p_v] - sequence_u[j]| > 1$. If v is not good in p_v, then it is a* bad node.

Lemma 69 states that once some node v becomes good in a period p_v, then its

2-hop neighbors always decide on round indexes corresponding to global rounds at least 3 rounds away from $g_v(i_v^e, p'_v)$ in all following periods p'_v. Thus, even with the jitter, v and its 2-hop neighbors always invoke BEEP2H at least 2 global rounds away, thus proving Lemma 70.

Lemma 69. *For any given node v, once v is good in some period p_v then for all periods $p'_v \geq p_v$: $\forall u \in \mathcal{N}_2(v)$, for any period $p_u \geq 1$ such that $g_u(start_u(p_u)) \geq g_v(start_v(p_v))$, $|g_v(i_v^e, p'_v) - g_v(i_u^e, p_u)| > 2$.*

Proof. For any node w and period $p_w \geq 2$, due to the properties of BEEP2H and LISTEN2H (from Section 7.2), the history of 2-hop communication during period $p_w - 1$ is complete at the end of the round indexed $4l$ of p_w. Then, by the definition of Algorithm 16 (line 24), for all periods $p'_v \geq p_v$, for any given node $u \in \mathcal{N}_2(v)$, node u (in some period $p_u \geq 1$ such that $g_u(start_u(p_u)) \geq g_v(start_v(p_v))$) chooses a round index i_u^e such that $|g_v(i_v^e, p'_v) - g_v(i_u^e, p_u)| > 2$. It is important to note that $i_u^e \leq T - 2$ (in every period), so that u knows whether neighbors beeped in the rounds indexed $T - 1$ and T of period $p_u - 1$ when deciding F_u in period p_u. □

Lemma 70. *For any given node v, once v is good in some period p_v then it remains good in all following periods $p'_v > p_v$ (even if 2-hop neighbors wake up in later periods).*

Once a node is good, it remains good forever. On the other hand, a bad node becomes good after two periods with constant probability (Lemma 74). Therefore, a bad node becomes a good node w.h.p. after $O(\log n)$ periods (Theorem 75). Lemma 74 is obtained by combining Lemmas 71 and 72. Lemma 72 builds upon Lemma 73, which proves that at least a constant fraction of the period is composed of free indexes.

Lemma 71. *Consider a bad node v in some period p_v with $disjoint_v(p_v + 1) =$ `true`. Then v is good in period $p_v + 1$ or $disjoint(p_v + 2) =$ `false` with probability at least $\frac{1}{2}$.*

Proof. Consider a bad node v in some period p_v with $disjoint_v(p_v + 1) =$ `true`. Since $disjoint_v(p_v + 1) =$ `true`, $i_v^e(p_v + 1) = i_v^e(p_v)$. Moreover, v is a bad node in p_v, so there exists a set of nodes $S \subset \mathcal{N}_2(v)$ such that for any node $u \in S$, there is a period p_u such that $|g_v(i_v^e(p_v), p_v) + jitter_v(p_v) - g_u(i_u^e(p_u), p_u) - jitter_u(p_u)| \leq 1$.

Consider a node u in S. If $disjoint_u(p_u + 1) =$ `false` then u chooses $i_u^e(p_u + 1)$ such that $|g_v(i_v^e(p_v + 1), p_v + 1) - g_u(i_u^e(p_u + 1), p_u + 1)| > 2$. Thus, if for all nodes u in S, $disjoint_u(p_u + 1) =$ `false` then v is good in period $p_v + 1$.
Otherwise, $\exists u \in S$ such that $disjoint_u(p_u + 1) =$ `true`. Thus $i_u^e(p_u + 1) = i_u^e(p_u)$. Since $jitter_v(p_v + 1)$ and $jitter_u(p_u + 1)$ are chosen from $\{0, 1\}$ with probability $\frac{1}{2}$, then $g_v(i_v^e(p_v + 1), p_v + 1) + jitter_v(p_v + 1) \neq g_u(i_u^e(p_u + 1), p_u + 1) + jitter_u(p_u + 1)$ with probability $\frac{1}{2}$. Since u and v beep in different rounds with probability $\frac{1}{2}$, $disjoint(p_v + 2) =$ `false` with probability at least $\frac{1}{2}$ (line 28). □

Lemma 72. *Consider a bad node v in some period p_v with $disjoint(p_v+1) =$ `false`. Then v is good in period $p_v + 1$ with probability at least $\exp \frac{-10}{(1-9\beta)\kappa}$.*

Lemma 73. *If $\kappa \geq 81$, then for any given node v and any given period $p_v \geq 2$ of v, $|F_v| \geq (1 - \frac{81}{\kappa})T$.*

Proof. Consider a node v and a period $p_v \geq 2$. Since indexes that are within 2 rounds of B are not in F_v, $|F_v| \geq T - (4l + 2) - 5(\Delta^2 + 1)$. As $l \leq \frac{16}{\kappa}T$, $|F_v| \geq T(1 - \frac{64}{\kappa} - 10\Delta^2 - 7)$. Following which, $|F_v| \geq T(1 - \frac{64}{\kappa} - \frac{17}{\kappa})$. Finally, $|F_v| \geq (1 - \frac{81}{\kappa})T$ (and $1 - \frac{81}{\kappa} \geq 0$ for $\kappa \geq 81$). □

Proof of Lemma 72. Consider a bad node v in some period p_v with $disjoint_v(p_v + 1) = \texttt{false}$. Since $disjoint_v(p_v) = \texttt{false}$, $i_v^e(p_v + 1)$ is chosen uniformly at random in $F_v(p_v + 1)$. Let $r' = g_v(i_v^e(p_v + 1), p_v + 1)$. Node v becomes good in $p_v + 1$ unless a non-empty subset of nodes $S \subset \mathcal{N}_2(v)$ invoke BEEP2H in $\{r' - 2, \ldots, r' + 2\}$ (at most once each). Taking into account the jitter, the probability p_u that a node $u \in S$ interferes with v is at most $\frac{6}{|F_u(p_u)|}$ for some period p_u, which is at most $\frac{6}{(1-\frac{81}{\kappa})T}$. Then, the probability that v is good in period $p_v + 1$ is $p = \prod_{u \in S}(1 - p_u) \geq (1 - p_u)^{|\mathcal{N}_2(v)|} \geq \exp \frac{-12}{(1-\frac{81}{\kappa})\kappa}$. The last inequality holds for $\frac{6}{(1-\frac{81}{\kappa})\kappa} \leq \frac{1}{2}$, thus $\kappa \geq 93$. □

Lemma 74. *Consider a bad node v in some period p_v. Then v is good in period $p_v + 2$ with constant probability.*

Theorem 75. *A bad node v becomes good after $O(\log n)$ periods with high probability.*

Concluding Remark. A node v executing Algorithm 16 decides w.h.p. on an arithmetic sequence of rounds $O(\log n)$ periods after its wake-up, such that no other awake node in its 2-hop neighborhood invokes BEEP2H within 2 rounds. During each following period, small intervals centered on the sequence's round (for that period) can be used to transmit bits. For any awake neighbor of v, v is the only node to invoke BEEP2H during these intervals, thus each bit is received without collisions. As a result the SEND and RECEIVE primitives, dealing with messages of any size, can be implemented on top of BEEP2H and LISTEN2H via this communication mechanism. These primitive are correct $O(\Delta_{up}^4 \log n)$ rounds after every node has woken up.

7.4 Summary

In a first part, we implement the *2-hop communication primitives* BEEP2H and LISTEN2H in $\mathcal{BEEP}_U$ (Section 7.2).

- To do so, we first introduce uncoordinated superimposed codes, which are an original combinatorial approach that we develop exploiting the properties of distinct difference sets (DDS) - see Section 7.2.1.
- Then, in Sections 7.2.2 and 7.2.3, we give algorithms for the 2-hop communication primitives BEEP2H and LISTEN2H building upon the technique of uncoordinated superimposed codes. These primitives replace BEEP and

LISTEN for communication on the square graph (i.e., nodes can communicate up to distance 2 with beeps or listenings). While being the crucial component of this work, these primitives are also quite general in the sense that they can be used for solving other problems (e.g., 2-hop MIS) in the beeping model with uncoordinated wake-ups.

In a second part, a solution to the *2-hop desynchronization problem* in $O(\Delta_{up}^4 \log n)$ rounds w.h.p. is presented in Section 7.3 (where Δ_{up} is some known upper bound on the maximum degree Δ). This result is particularly significant since it allows to implement the higher level communication primitives SEND and RECEIVE as in the message passing model. This in turn allows to synchronize the local clocks, desynchronized because of the arbitrary wake-ups.

Chapter 8

Conclusion

8.1 Overview

Algorithm design in the beeping model is challenging. Indeed, beeps are simple unary signals, and suffer from some degree of information loss when collisions happen. Our results provide interference control, using symmetry-breaking primitives and coding techniques. Importantly, efficient interference control allows for efficient communication primitives in the beeping model.

On the one hand, we addressed interference control on a local scale. First, we examined local symmetry-breaking problems - vertex coloring and maximal independent set (Chapter 4) - in the synchronous starts setting (i.e., $\mathcal{BEEP}_S$). Deterministic and uniform solutions for these problems were proposed, as well as solutions for their 2-hop variants. Using these, message-passing between neighboring nodes can be simulated in exchange for some overhead. Following which, we investigated a different symmetry-breaking problem - 2-hop desynchronization (Chapter 7) - in the uncoordinated starts setting (i.e., $\mathcal{BEEP}_U$). As a first step, we introduced an original coding technique to obtain the first 2-hop communication primitive. Using this, we presented the first (randomized) 2-hop desynchronization algorithm. Importantly, this solution can also be used to implement message-passing with some overhead, but in the more practically-relevant uncoordinated starts setting.

On the other hand, we have also addressed interference control on a global scale. First, we studied the leader election problem (Chapter 5) - a global symmetry-breaking problem - and designed a time-optimal deterministic and uniform algorithm in $\mathcal{BEEP}_S$. Then, we focused on a fundamental communication primitive: multi-broadcast (Chapter 6). Building upon the efficient leader election solution (from Chapter 5) and coding techniques, computationally- and time-efficient solutions for multi-broadcast were presented. The solutions given in Chapters 5 and 6 show that on a global scale, dealing with non-destructive interference can be done without any negative impact (i.e., asymptotic overhead).

In conclusion, the results of the thesis shed some light on how the degree of synchronization between nodes can impact the design of efficient (interference control) solutions in the beeping model. On the one hand, in the synchronous starts setting, in which nodes have synchronized local clocks, time-efficient *deterministic* and *uniform* solutions were obtained (in Chapters 4 to 6). For these solutions, techniques that leverage the strong degree of synchronization to convey, using beeps, small amounts of collision-tolerant information, are crucial. On the other hand, in the uncoordinated starts setting, nodes can have arbitrarily different local clocks and as a result, such techniques are unavailable. Instead, to cope with the lack of synchronization, we introduce an original coding technique (in Chapter 7). How-

ever, some knowledge on graph parameters (resulting in a *non-uniform* solutions), as well as *randomization*, are required. Whether this is necessary in such a harsh setting is an open question. This question, and other questions related to the thesis, are addressed below.

8.2 Perspectives

We start by discussing perspectives following directly from the results of the thesis. First, notice that while the proposed global interference control methods are efficient and have no negative impact (asymptotically), that is not the case for the local interference control methods. Can local interference control be achieved even more efficiently in the beeping model? How much (if any) of a negative impact does optimal local interference control produce?

Furthermore, an important result of this thesis shows that local interference control is also possible in the uncoordinated starts setting. Since most previous works have considered the synchronous starts setting (see the related work in Chapter 3), the uncoordinated starts setting is still poorly understood. In particular, how do beeps efficiently and reliably convey information in this setting? Can they even do so in a uniform manner (i.e., without any parameter knowledge or purely deterministically)? Finally, can efficient global interference control be achieved in this setting?

Now we take a step back and propose three different future lines of research.

Difference between Beeps and 1-bit Messages. Recent works study the impact of reducing the message size in traditional message-passing models. [74, 24] consider $\mathcal{CONGEST}$ limited to 1 bit messages, and respectively present coloring and leader election solutions. The beeping model is closely related to this model. $\mathcal{CONGEST}$ with 1-bit messages can simulate beeping algorithms, but it is yet unclear if the beeping model is weaker, and if yes, to what extent it is so. By studying how communicating by beeps impacts algorithmic solutions, it is possible to understand if and how a stronger communication is required for efficient algorithms.

In this work, it is shown that for both the beeping model (with synchronous starts) and $\mathcal{CONGEST}$ with 1-bit messages, $O(D + \log n)$ solutions for leader election can be obtained. Additionally, [42] shows that multi-broadcast can also be solved with the same asymptotical complexity in both models (if computation efficiency is not taken into account).

However, what about local scale (symmetry-breaking) problems? Is there a gap between both models for the coloring and MIS problems? As an intermediate step, problems providing a smaller degree of symmetry-breaking (e.g., ruling set) can also be taken into account.
Moreover, local scale symmetry-breaking problems allow to simulate $\mathcal{CONGEST}$ communication. In this work, we provide $\mathcal{CONGEST}$ simulations in both the synchronous starts and uncoordinated starts setting. However, these simulations (even for 1-bit messages) have high multiplicative overheads that depend on the communication graph's maximum degree. Then, is it possible to design simulations with

constant overhead (possibly by obtaining better coloring solutions)?

Fault-Tolerance in the Beeping Model. Electronic devices have become more and more widespread, at the cost of reduced reliability. In other words, these devices suffer more often from faults, e.g., crashes or transient faulty behaviors, due to mass-production and large-scale deployment. Self-stabilizing distributed algorithms provide solutions that handle transient faults in a distributed manner. Classical self-stabilizing solution generally rely on perpetually communicating large amounts of information [4] in order to detect and correct the inconsistencies produced by transient faults. However, it is yet unknown whether such perpetual information exchange can be obtained under transient faults using beeps. As such, obtaining self-stabilization solutions in the beeping model may only be possible for problems in which inconsistencies can be easily detected using beeps. For instance, for the MIS and desynchronization problems, a simple probabilistic collision detection primitive (using randomization) can detect inconsistencies, and randomized self-stabilizing solutions have been given [2, 38]. On the other hand, detecting inconsistencies in problems such as clock synchronization and vertex coloring is more challenging. Can self-stabilizing solutions for these problems be obtained in the uncoordinated starts setting? Moreover, is randomization necessary to ensure self-stabilization in the beeping model?

Furthermore, with the rise of the Internet of Things, security issues have taken the forefront. Solutions that deal with byzantine faults [77], which model arbitrary non-mobile faulty behaviors, allow to cope with some of these issues. However, byzantine faults appear to be very difficult to tackle in the beeping model. In particular, byzantine fault masking may not be possible, that is, it may not be possible for all correct nodes to have correct outputs. Indeed, although a byzantine node can transmit very little false information using a beep, it can choose to beep in all rounds. In doing so, its neighbors are unable to obtain any information. To deal with that, [64] augments the beeping model with the ability to count the number of beeping nodes, thus allowing nodes to handle byzantine faults.

Nevertheless, instead of byzantine fault masking, is byzantine containment [87] achievable in the original beeping model? More precisely, can nodes contain byzantine faults, such that correct nodes that are at a certain distance away from a byzantine node are able to give a correct output?

Beeps, Fireflies and Synchronization. Among emerging phenomena in nature, the synchronized flashing phenomenon of firefly swarms [21] offers a riveting display of persistent regular behavior achievable in a distributed system with limited communication. Such synchronization behaviors are yet to be understood fully. Improving our knowledge regarding these behaviors has implications from a purely scientific perspective, but also from a practical perspective. Indeed, synchronization solutions allow nodes to synchronize their local clocks, which they can then leverage for more complex solutions (as has been show for the beeping model in this work).

Beeps offer severely limited communication capabilities. As a result, several parallels can be drawn between the distributed synchronization behavior in firefly swarms and the study of the synchronization problem in the beeping model. [6] gives

a synchronization solution for the wake-on-beep setting, whereas [64] considers self-stabilizing synchronization with byzantine faults, but augments the beeping model with the capability to count the number of neighboring beeping nodes. However, it would be interesting to consider the synchronization problem in the uncoordinated starts setting (i.e., $\mathcal{BEEP}_U$), for which both works give little intuition. In this harsh setting, can synchronization be achieved using only beeps, or is it necessary to assume some stronger capabilities, e.g., being able to count the number of beeping nodes? In a distributed system relying on beeps for communication, does synchronization require the use of randomization?

Bibliography

[1] Y. Afek, N. Alon, Z. Bar-Joseph, A. Cornejo, B. Haeupler, and F. Kuhn. Beeping a maximal independent set. In *Proceedings of the 25th International Symposium on Distributed Computing (DISC)*, pages 32–50, 2011.

[2] Y. Afek, N. Alon, Z. Bar-Joseph, A. Cornejo, B. Haeupler, and F. Kuhn. Beeping a maximal independent set. *Distributed Computing*, 26(4):195–208, 2013.

[3] Y. Afek, N. Alon, O. Barad, E. Hornstein, N. Barkai, and Z. Bar-Joseph. A biological solution to a fundamental distributed computing problem. *Science*, 331(6014):183–185, 2011.

[4] Y. Afek, S. Kutten, and M. Yung. The local detection paradigm and its applications to self-stabilization. *Theoretical Computer Science*, 186(1):199 – 229, 1997.

[5] G. Agnarsson and M.M. Halldorsson. Coloring powers of planar graphs. *SIAM Journal on Discrete Mathematics*, 16(4):651–662, 2003.

[6] D. Alistarh, A. Cornejo, M. Ghaffari, and N. Lynch. Firefly synchronization with asynchronous wake-up. In *Workshop on Biological Distributed Algorithms*, 2014.

[7] D. Angluin. Local and global properties in networks of processors (extended abstract). In *Proceedings of the 12th Annual ACM Symposium on Theory of Computing (STOC)*, pages 82–93, 1980.

[8] D. Angluin, J. Aspnes, Z. Diamadi, M. J. Fischer, and R. Peralta. Computation in networks of passively mobile finite-state sensors. *Distributed Computing*, 18(4):235–253, March 2006.

[9] B. Awerbuch, M. Luby, A. V. Goldberg, and S. A. Plotkin. Network decomposition and locality in distributed computation. In *Proceedings of the 30th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 364–369, 1989.

[10] R. Bar-Yehuda, A. Israeli, and A. Itai. Multiple communication in multihop radio networks. *SIAM Journal on Computing*, 22(4):875–887, 1993.

[11] L. Barenboim and M. Elkin. Sublogarithmic distributed mis algorithm for sparse graphs using nash-williams decomposition. In *Proceedings of the 27th Annual ACM Symposium on Principles of Distributed Computing (PODC)*, pages 363–379, 2008.

[12] L. Barenboim and M. Elkin. Distributed (δ+1)-coloring in linear (in δ) time. In *Proceedings of the 41st Annual ACM Symposium on Theory of Computing (STOC)*, pages 111–120, 2009.

[13] L. Barenboim and M. Elkin. Deterministic distributed vertex coloring in polylogarithmic time. In *Proceedings of the 29th Annual ACM Symposium on Principles of Distributed Computing (PODC)*, pages 410–419, 2010.

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

[15] J. Beauquier, J. Burman, P. Davies, and F. Dufoulon. Optimal multi-broadcast with beeps using group testing. In *Proceedings of the 26th International Colloquium on Structural Information and Communication Complexity (SIROCCO)*, pages 65–79, 2019.

[16] J. Beauquier, J. Burman, F. Dufoulon, and S. Kutten. Fast Beeping Protocols for Deterministic MIS and (Δ+1)-Coloring in Sparse Graphs. In *Proceedings of the 37th IEEE Conference on Computer Communications (INFOCOM)*, pages 1754–1762, 2018.

[17] P. Błaśkiewicz, M. Klonowski, M. Kutyłowski, and P. Syga. Lightweight protocol for trusted spontaneous communication. In *Proceedings of the 6th International Conference on Trusted Systems (INTRUST)*, pages 228–242, 2014.

[18] A. Bonis, L. Gasieniec, and U. Vaccaro. Optimal two-stage algorithms for group testing problems. *SIAM Journal on Computing*, 34(5):1253–1270, 2005.

[19] P. Brandes, M. Kardas, M. Klonowski, D. Pajak, and R. Wattenhofer. Approximating the size of a radio network in beeping model. In *Proceedings of the 23rd International Colloquium on Structural Information and Communication Complexity (SIROCCO)*, pages 358–373, 2016.

[20] S. J. Bray. Notch signaling: a simple pathway becomes complex. *Nature Reviews Molecular Cell Biology*, 7:678–689, 10 2006.

[21] John Buck. Synchronous rhythmic flashing of fireflies. ii. *The Quarterly Review of Biology*, 63(3):265–289, 1988.

[22] A. Casteigts, P. Flocchini, W. Quattrociocchi, and N. Santoro. Time-varying graphs and dynamic networks. *International Journal of Parallel, Emergent and Distributed Systems*, 27(5):387–408, 2012.

[23] A. Casteigts, Y. Métivier, J. M. Robson, and A. Zemmari. Design patterns in beeping algorithms. In *Proceedings of the 20th International Conference on Principles of Distributed Systems (OPODIS)*, pages 15:1–15:16, 2016.

[24] A. Casteigts, Y. Métivier, J. M. Robson, and A. Zemmari. Deterministic leader election takes Θ(d+log n) bit rounds. *Algorithmica*, 81(5):1901–1920, 2019.

[25] A. Casteigts, Y. Métivier, J.M. Robson, and A. Zemmari. Deterministic leader election in $\mathcal{O}(D + \log n)$ time with messages of size $\mathcal{O}(1)$. In *Proceedings of*

the 30th International Symposium on Distributed Computing (DISC), pages 16–28, 2016.

[26] A. Casteigts, Y. Métivier, J.M. Robson, and A. Zemmari. Counting in one-hop beeping networks. *Theoretical Computer Science*, 2019.

[27] A. Casteigts, Y. Métivier, J.M. Robson, and A. Zemmari. Design patterns in beeping algorithms: Examples, emulation, and analysis. *Information and Computation*, 264:32 – 51, 2019.

[28] P. Chebyshev. Mémoire sur les nombres premiers. *Journal de mathématiques pures et appliquées* 1^{re} *série*, 17:366–390, 1852.

[29] M. Cheraghchi. Noise-resilient group testing: Limitations and constructions. *Discrete Applied Mathematics*, 161(1):81 – 95, 2013.

[30] B. S. Chlebus, G. De Marco, and M. Talo. Naming a channel with beeps. *Fundamenta Informaticae*, 153:199–219, 2017.

[31] B. S. Chlebus, L. Gąsieniec, D. R. Kowalski, and T. Radzik. On the wake-up problem in radio networks. In *Proceedings of the 32nd International Colloquium on Automata, Languages and Programming (ICALP)*, pages 347–359, 2005.

[32] B. S. Chlebus and D. R. Kowalski. A better wake-up in radio networks. In *Proceedings of the 23rd Annual ACM Symposium on Principles of Distributed Computing (PODC)*, pages 266–274, 2004.

[33] B. S. Chlebus, D. R. Kowalski, A. Pelc, and M. A. Rokicki. Efficient distributed communication in ad-hoc radio networks. In *Proceedings of the 38th International Colloquium on Automata, Languages and Programming (ICALP)*, pages 613–624, 2011.

[34] M. Chrobak, L. Gąsieniec, and D. Kowalski. The wake-up problem in multi-hop radio networks. *SIAM Journal on Computing*, 36(5):1453–1471, 2007.

[35] F. Cicalese and U. Vaccaro. Superselectors: Efficient constructions and applications. In *Proceedings of the 18th Annual European Symposium of Algorithms (ESA)*, pages 207–218, 2010.

[36] R. Cole and U. Vishkin. Deterministic coin tossing with applications to optimal parallel list ranking. *Information and Control*, 70(1):32 – 53, 1986.

[37] J. R. Collier, N. A. M. Monk, P. K. Maini, and J. H. Lewis. Pattern formation by lateral inhibition with feedback: a mathematical model of delta-notch intercellular signalling. *Journal of Theoretical Biology*, 183(4):429–446, 1996.

[38] A. Cornejo and F. Kuhn. Deploying wireless networks with beeps. In *Proceedings of the 24th International Symposium on Distributed Computing (DISC)*, pages 148–162, 2010.

[39] A. Czumaj and P. Davies. Brief announcement: Optimal leader election in multi-hop radio networks. In *Proceedings of the 35th Annual ACM Symposium on Principles of Distributed Computing (PODC)*, pages 47–49, 2016.

[40] A. Czumaj and P. Davies. Communicating with Beeps. In *Proceedings of the 19th International Conference on Principles of Distributed Systems (OPODIS)*, pages 1–16, 2016.

[41] A. Czumaj and P. Davies. Deterministic communication in radio networks. *SIAM Journal on Computing*, 47(1):218–240, 2018.

[42] A. Czumaj and P. Davies. Communicating with beeps. *Journal of Parallel and Distributed Computing*, 130:98 – 109, 2019.

[43] J. Degesys and R. Nagpal. Towards desynchronization of multi-hop topologies. In *Proceedings of the 2nd IEEE International Conference on Self-Adaptive and Self-Organizing Systems (SASO)*, pages 129–138, 2008.

[44] J. Degesys, I. Rose, A. Patel, and R. Nagpal. Desync: Self-organizing desynchronization and tdma on wireless sensor networks. In *Proceedings of the 6th International Conference on Information Processing in Sensor Networks (ISPN)*, pages 11–20, 2007.

[45] Edsger W. Dijkstra. Self-stabilizing systems in spite of distributed control. *Communications of the ACM*, 17(11):643–644, November 1974.

[46] A. Dimitromanolakis. Analysis of the Golomb ruler and the Sidon set problems, and determination of large, near-optimal Golomb rulers. Master's thesis, Department of Electronic and Computer Engineering, Technical University of Crete, 2002.

[47] Y. Dinitz and N. Solomon. Two absolute bounds for distributed bit complexity. In *Proceedings of the 12th International Colloquium on Structural Information and Communication Complexity (SIROCCO)*, pages 115–126, 2005.

[48] Robert Dorfman. The detection of defective members of large populations. *Ann. Math. Statist.*, 14(4):436–440, 1943.

[49] D-Z. Du and F. K. Hwang. *Combinatorial Group Testing and Its Applications*. World Scientific, 1993.

[50] F. Dufoulon, J. Burman, and J. Beauquier. Solving 2 hop desynchronization in the beeping model. Submitted.

[51] F. Dufoulon, J. Burman, and J. Beauquier. Beeping a deterministic time-optimal leader election. In *Proceedings of the 32nd International Symposium on Distributed Computing (DISC)*, pages 20:1–20:17, 2018.

[52] F. Dufoulon, J. Burman, and J. Beauquier. Brief announcement: Beeping a time-optimal leader election. In *Proceedings of the 37th Annual ACM Symposium on Principles of Distributed Computing (PODC)*, pages 237–239, 2018.

[53] A.G. D'yachkov, V.V. Rykov, and A.M. Rashad. Superimposed distance codes. *Problems of Control and Information Theory*, 18(4):237–250, 1989.

[54] S. Elouasbi and A. Pelc. Deterministic rendezvous with detection using beeps. In *Proceedings of the 11th International Symposium on Algorithms and Experiments for Wireless Sensor Networks (ALGOSENSORS)*, pages 85–97, 2015.

[55] Y. Emek and R. Wattenhofer. Stone age distributed computing. In *Proceedings of the 32nd Annual ACM Symposium on Principles of Distributed Computing (PODC)*, pages 137–146, 2013.

[56] D. Eppstein and D. Strash. Listing all maximal cliques in large sparse real-world graphs. In *Proceedings of the 10th International Symposium on Experimental Algorithms (SEA)*, pages 364–375, 2011.

[57] O. Feinerman and A. Korman. Theoretical distributed computing meets biology: A review. In *Distributed Computing and Internet Technology*, pages 1–18, 2013.

[58] K.-T. Förster, J. Seidel, and R. Wattenhofer. Deterministic leader election in multi-hop beeping networks. In *Proceedings of the 28th International Symposium on Distributed Computing (DISC)*, pages 212–226, 2014.

[59] L. Gąsieniec, A. Pelc, and D. Peleg. The wakeup problem in synchronous broadcast systems. *SIAM Journal on Discrete Mathematics*, 14(2):207–222, 2001.

[60] M. Ghaffari and B. Haeupler. Near optimal leader election in multi-hop radio networks. In *Proceedings of the 24th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 748–766, 2013.

[61] S. Gilbert and C. Newport. The computational power of beeps. In *Proceedings of the 29th International Symposium on Distributed Computing (DISC)*, pages 31–46, 2015.

[62] S. Gilbert and C. Newport. Symmetry breaking with noisy processes. In *Proceedings of the 36th Annual ACM Symposium on Principles of Distributed Computing (PODC)*, pages 273–282, 2017.

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

[64] R. Guerraoui and A. Maurer. Byzantine fireflies. In *Proceedings of the 29th International Symposium on Distributed Computing (DISC)*, pages 47–59, 2015.

[65] M. Halldórsson and C. Konrad. Computing large independent sets in a single round. *Distributed Computing*, 31(1):69–82, 2018.

[66] A. J. Hoffman, K. Jenkins, and T. Roughgarden. On a game in directed graphs. *Information Processing Letters*, 83(1):13–16, 2002.

[67] S. Holzer and N. Lynch. Brief announcement: Beeping a maximal independent set fast. In *Proceedings of the 30th International Symposium on Distributed Computing (DISC)*, 2016.

[68] K. Hounkanli, A. Miller, and A. Pelc. Global synchronization and consensus using beeps in a fault-prone mac. In *Proceedings of the 12th International Symposium on Algorithms and Experiments for Wireless Sensor Networks (ALGOSENSORS)*, pages 16–28, 2017.

[69] K. Hounkanli and A. Pelc. Asynchronous broadcasting with bivalent beeps. In *Proceedings of the 23rd International Colloquium on Structural Information and Communication Complexity (SIROCCO)*, pages 291–306, 2016.

[70] B. Huang and T. Moscibroda. Conflict resolution and membership problem in beeping channels. In *Proceedings of the 27th International Symposium on Distributed Computing (DISC)*, pages 314–328, 2013.

[71] F. K. Hwang. A method for detecting all defective members in a population by group testing. *Journal of the American Statistical Association*, 67(339):605–608, 1972.

[72] P. Indyk, H. Q. Ngo, and A. Rudra. Efficiently decodable non-adaptive group testing. In *Proceedings of the 21st Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1126–1142, 2010.

[73] W. H. Kautz and R. C. Singleton. Nonrandom binary superimposed codes. *IEEE Transaction on Information Theory*, 10(4):363–377, 1964.

[74] K. Kothapalli, C. Scheideler, M. Onus, and C. Schindelhauer. Distributed coloring in $\tilde{O}(\sqrt{\log N})$ bit rounds. In *Proceedings of the 20th International Conference on Parallel and Distributed Processing*, IPDPS'06, 2006.

[75] F. Kuhn. Weak graph colorings: Distributed algorithms and applications. In *Proceedings of the 21st Annual Symposium on Parallelism in Algorithms and Architectures (SPAA)*, pages 138–144, 2009.

[76] S. Kutten, G. Pandurangan, D. Peleg, P. Robinson, and A. Trehan. On the complexity of universal leader election. In *Proceedings of the 32nd Annual ACM Symposium on Principles of Distributed Computing (PODC)*, pages 100–109, 2013.

[77] L. Lamport, R. Shostak, and M. Pease. The byzantine generals problem. *ACM Transactions on Programming Languages and Systems*, pages 382–401, July 1982.

[78] C. H. Li. A sequential method for screening experimental variables. *Journal of the American Statistical Association*, 57(298):455–477, 1962.

[79] Z. Liu and M. Herlihy. Approximate local sums and their applications in radio networks. In *Proceedings of the 28th International Symposium on Distributed Computing (DISC)*, pages 243–257, 2014.

[80] J. Luo and D. Guo. Neighbor discovery in wireless ad hoc networks based on group testing. In *46th Annual Allerton Conference on Communication, Control, and Computing*, pages 791–797, 2008.

[81] Y. Métivier, J.M. Robson, and A. Zemmari. Analysis of fully distributed splitting and naming probabilistic procedures and applications. *Theoretical Computer Science*, 584:115 – 130, 2015. Special Issue on Structural Information and Communication Complexity.

[82] A. Motskin, T. Roughgarden, P. Skraba, and L. Guibas. Lightweight coloring and desynchronization for networks. In *Proceedings of the 28th IEEE Conference on Computer Communications (INFOCOM)*, pages 2383–2391, 2009.

[83] K. Nakano and S. Olariu. Randomized $O(\log \log n)$-round leader election protocols in packet radio networks. In *Proceedings of the 9th International Symposium on Algorithms and Computation (ISAAC)*, pages 210–219, 1998.

[84] C. St. J. A. Nash-Williams. Decomposition of finite graphs into forests. *Journal of the London Mathematical Society*, 39:12, 1964.

[85] S. Navlakha and Z. Bar-Joseph. Algorithms in nature: the convergence of systems biology and computational thinking. *Molecular Systems Biology*, 7(1), 2011.

[86] S. Navlakha and Z. Bar-Joseph. Distributed information processing in biological and computational systems. *Communications of the ACM*, 58(1):94–102, December 2014.

[87] M. Nesterenko and A. Arora. Tolerance to unbounded byzantine faults. In *Proceedings of the 21st IEEE Symposium on Reliable Distributed Systems (SRDS)*, pages 22–29, 2002.

[88] H. Q. Ngo, E. Porat, and A. Rudra. Efficiently decodable error-correcting list disjunct matrices and applications. In *Proceedings of the 38th International Colloquium on Automata, Languages and Programming (ICALP)*, pages 557–568, 2011.

[89] R. Oshman. *Distributed Computation in Wireless and Dynamic Networks.* PhD thesis, Massachusetts Institute of Technology, 2012.

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

[91] E. Porat and A. Rothschild. Explicit nonadaptive combinatorial group testing schemes. *IEEE Transactions on Information Theory*, 57(12):7982–7989, 2011.

[92] C. Scheideler, A. Richa, and P. Santi. An $O(\log n)$ dominating set protocol for wireless ad-hoc networks under the physical interference model. In *Proceedings of the International Symposium on Mobile Ad Hoc Networking and Computing (MobiHoc)*, pages 91–100, 2008.

[93] J. Schneider, M. Elkin, and R. Wattenhofer. Symmetry breaking depending on the chromatic number or the neighborhood growth. *Theoretical Computer Science*, 509(C):40–50, October 2013.

[94] J. Schneider and R. Wattenhofer. What is the use of collision detection (in wireless networks)? In *Proceedings of the 24th International Symposium on Distributed Computing (DISC)*, pages 133–147, 2010.

[95] J. Schneider and R. Wattenhofer. Distributed coloring depending on the chromatic number or the neighborhood growth. In *Proceedings of the 18th International Colloquium on Structural Information and Communication Complexity (SIROCCO)*, pages 246–257, 2011.

[96] A. Scott, P. Jeavons, and L. Xu. Feedback from nature: An optimal distributed algorithm for maximal independent set selection. In *Proceedings of the 32nd Annual ACM Symposium on Principles of Distributed Computing (PODC)*, pages 147–156, 2013.

[97] J. Seidel. *Anonymous distributed computing: computability, randomization and checkability*. PhD thesis, ETH Zurich, Zürich, Switzerland, 2015.

[98] J. Singer. A theorem in finite projective geometry and some applications to number theory. *Transactions of the American Mathematical Society*, 43(3):377–385, 1938.

[99] Salil P. Vadhan. Pseudorandomness. *Foundations and Trends in Theoretical Computer Science*, 7(1–3):1–336, 2012.

[100] J. Yu, L. Jia, D. Yu, G. Li, and X. Cheng. Minimum connected dominating set construction in wireless networks under the beeping model. In *Proceedings of the 34th IEEE Conference on Computer Communications (INFOCOM)*, pages 972–980, 2015.

Annexe A

Synthèse

Les petits appareils électroniques peu coûteux et à communication sans fil sont devenus largement disponibles. Bien que chaque entité ait des capacités limitées (par exemple, communication basique ou mémoire de taille constante), un déploiement à grande échelle de telles entités communiquantes constitue un réseau performant, en plus d'être peu coûteux. De tels systèmes distribués présentent toutefois des défis importants en ce qui concerne la conception d'algorithmes simples, efficaces et évolutifs.

Dans cette thèse, nous nous intéressons à l'étude de ces systèmes, composés d'appareils dotés de *capacités de communication très limitées*, à base de simples impulsions d'énergie. Ces systèmes distribués peuvent être modélisés à l'aide du *modèle de bips*, dans lequel les noeuds communiquent en émettant un bip, un simple signal indifférencié, ou en écoutant leurs voisins (selon un graphe de communication non orienté). Les communications simultanées (c'est-à-dire les *collisions*) entraînent des *interférences non destructives* : un noeud, dont deux voisins ou plus émettent simultanément un bip, détecte seulement un bip. Ce mécanisme de communication simple, général et économe en énergie rend les résultats obtenus dans le modèle de bips applicables à de nombreuses situations différentes, avec cependant un challenge. En raison de la faible expressivité des bips et des collisions, la conception des algorithmes est difficile. Tout au long de ce travail, nous surmontons ces deux difficultés afin de fournir des primitives de communication efficaces. La thèse s'intéresse particulièrement aux solutions déterministes, rapides (en temps) et indépendantes des paramètres du graphe de communication (c'est-à-dire uniformes).

A.1 Réveils Synchrones des noeuds

La première partie de la thèse considère un cadre dans lequel les noeuds se réveillent en même temps (c'est-à-dire que le réseau a été configuré a priori). La majorité des résultats obtenus dans le modèle de bips suppose ce cadre.

Pour obtenir des solutions efficaces pour des *problèmes fondamentaux de communication distribuée*, nous nous concentrons d'abord sur la résolution efficace de problèmes de brisure *locale* de symétrie : *ensemble indépendant maximal* et *coloration de sommets* utilisant au plus $\Delta + 1$ couleurs (où Δ est le degré maximal du graphe de communication). Ce sont deux problèmes majeurs en algorithmique distribuée. Un ensemble indépendant maximal permet une décomposition du réseau en petits groupes (de rayon 1), chacun dirigé par un noeud de l'ensemble. Une telle décomposition peut être utilisée pour la répartition des tâches et le partage

des ressources dans le réseau. Quant à la coloration, ce problème est un composant fondamental dans les réseaux à communications radios, et plus généralement les réseaux sans fil. En effet, les couleurs (des noeuds) peuvent servir d'identifiants locaux et permettent donc une brisure de symétrie locale, essentielle dans la gestion d'interférence et la résolution de problèmes distribués plus complexes.

Nous élaborons des solutions à ces problèmes ainsi qu'à leurs variantes à distance deux. Cela nous permet de simuler une communication par messages. Ces solutions sont particulièrement efficaces lorsque le graphe de communication est peu dense. Enfin, en combinant avec certains résultats existants, qui s'appuient sur la communication de messages, nous obtenons le premier algorithme de coloration utilisant moins de $\Delta + 1$ couleurs dans le modèle de bips.

Ensuite, nous étudions des problèmes définis à l'échelle du réseau, tels que l'*élection d'un leader* et la *diffusion multiple* de messages. L'élection d'un leader est un élément essentiel dans la conception d'algorithmes distribués. En effet, un leader dispose d'une autorité absolue sur le réseau et peut donc coordonner les autres noeuds pour limiter les interférences à l'échelle du réseau. Il peut aussi démarrer la construction d'un arbre couvrant afin de récupérer efficacement l'information de l'ensemble du réseau. Ceci est un premier pas, crucial, dans la diffusion multiple de messages, où plusieurs noeuds sources cherchent à communiquer leur message (et possiblement leur identifiant) à tous les noeuds du réseaux, de façon efficace.

Nous donnons les deux premiers algorithmes d'élection de leader optimaux en temps pour le modèle de bips. L'un est déterministe, mais nécessite des identifiants uniques. Le second n'a pas besoin d'identifiants (utile pour des raisons de sécurité et de confidentialité), mais est randomisé. S'appuyant sur une élection de leader optimale en temps, plusieurs algorithmes pour la diffusion multiple, efficaces en temps et en calcul, sont présentés. Bien qu'une solution précédente (pour la diffusion multiple), optimale en temps, soit disponible, elle nécessite des méthodes coûteuses en calcul.

A.2 Réveils asynchrones des noeuds

La deuxième partie de la thèse considère un cadre plus difficile mais plus général, dans lequel les noeuds se réveillent de façon asynchrone. La conception de solutions dans ce cadre est d'importance majeur : elles pourront servir de point d'appui pour obtenir des solutions tolérantes aux fautes transitoires ou aux réseaux dynamiques (dans lequel les noeuds se joignent au réseau, ou le quittent, de façon arbitraire).

Nous nous concentrons sur le problème de *désynchronisation à distance deux*, qui permet un contrôle de l'accès au support, primordial dans les réseaux sans fil. Dans ce problème, les noeuds cherchent à émettre des bips de façon périodique, tout en évitant d'émettre dans la même ronde qu'un autre noeud voisin ou à distance deux : c'est-à-dire, de façon désynchronisée à distance deux. Cela nécessite que les noeuds transmettent et recoivent de l'information dans leur voisinage à distance deux. Puisqu'un bip ne se transmet qu'aux voisins, communiquer dans le voisinage à distance

deux nécessite qu'un noeud se coordonne avec ses voisins pour qu'ils retransmettent ensuite l'information à leurs voisins. Ce type de problématique n'avait pas été considéré auparavant et démarque notre travail des résultats précédents dans le modèle de bips avec réveils asynchrones.

Pour élaborer une solution au problème de désynchronisation à distance deux, nous montrons dans un premier temps qu'il est possible pour les noeuds de communiquer de manière cohérente au-delà de leur voisinage immédiat, en utilisant des outils de la théorie du code. À cette fin, une primitive permettant aux noeuds de simuler une communication sur le carré du graphe de communication est présentée. Cette primitive est un élément central dans la conception de l'algorithme de désynchronisation à distance deux, et s'appuie sur un type de code, non considéré jusque là, pour lequel nous donnons une construction originale.

Dans un deuxième temps, nous montrons précisément comment cette primitive peut être composée avec un mécanisme probabiliste, pour désynchroniser les noeuds à distance deux. Celle-ci permet un contrôle de l'accès au support, afin d'implémenter des primitives de haut niveau pour l'envoi et la réception de messages. Ces primitives, ayant été conçues pour des réseaux sans fil communiquants à travers des bips, sont extrêmement générales.

université PARIS-SACLAY
ÉCOLE DOCTORALE
Sciences et technologies de l'information et de la communication (STIC)

Titre : Surmonter les interférences dans le modèle de communication par bips

Mots clés : modèle de bips, contrôle des interférences, réseaux sans fil, appareils électroniques élémentaires, communication basique

Résumé : Les petits appareils électroniques peu coûteux et à communication sans fil sont devenus largement disponibles. Bien que chaque entité ait des capacités limitées (par exemple, communication basique ou mémoire de taille constante), un déploiement à grande échelle de telles entités communiquantes constitue un réseau performant, en plus d'être peu coûteux. De tels systèmes distribués présentent toutefois des défis importants en ce qui concerne la conception d'algorithmes simples, efficaces et évolutifs.

Dans cette thèse, nous nous intéressons à l'étude de ces systèmes, composés d'appareils dotés de *capacités de communication très limitées*, à base de simples impulsions d'énergie. Ces systèmes distribués peuvent être modélisés à l'aide du *modèle de bips*, dans lequel les noeuds communiquent en émettant un bip, un simple signal indifférencié, ou en écoutant leurs voisins (selon un graphe de communication non orienté). Les communications simultanées (c'est-à-dire les *collisions*) entraînent des *interférences non destructives* : un noeud, dont deux voisins ou plus émettent simultanément un bip, détecte seulement un bip. Ce mécanisme de communication simple, général et économe en énergie rend les résultats obtenus dans le modèle de bips applicables à de nombreuses situations différentes, avec cependant un challenge. En raison de la faible expressivité des bips et des collisions, la conception des algorithmes est difficile. Tout au long de ce travail, nous surmontons ces deux difficultés afin de fournir des primitives de communication efficaces.

La thèse s'intéresse particulièrement aux solutions déterministes, rapides (en temps) et indépendantes des paramètres du graphe de communication (c'est-à-dire uniformes).

La première partie de la thèse considère un cadre dans lequel les noeuds se réveillent en même temps (c'est-à-dire que le réseau a été configuré a priori). Pour obtenir des solutions efficaces pour des *problèmes fondamentaux de communication distribuée*, nous nous concentrons d'abord sur la résolution efficace de problèmes de brisure *locale* de symétrie : *ensemble indépendant maximal* et *coloration de sommets* utilisant au plus $\Delta + 1$ couleurs (où Δ est le degré maximal du graphe de communication). Nous élaborons des solutions à ces problèmes ainsi qu'à leurs variantes à distance deux. Cela nous permet de simuler une communication par messages. Enfin, nous obtenons le premier algorithme de coloration utilisant moins de $\Delta + 1$ couleurs dans le modèle de bips. Ensuite, nous étudions des problèmes définis à l'échelle du réseau, tels que l'*élection d'un leader* et la *diffusion multiple* de messages. L'élection d'un leader est un élément essentiel dans la conception d'algorithmes distribués. Nous donnons les deux premiers algorithmes d'élection de leader optimaux en temps pour le modèle de bips. L'un est déterministe, mais nécessite des identifiants uniques. Le second n'a pas besoin d'identifiants (utile pour des raisons de sécurité et de confidentialité), mais est randomisé. S'appuyant sur une élection de leader optimale en temps, plusieurs algorithmes pour la diffusion multiple, efficaces en temps et en calcul, sont présentés.

La deuxième partie de la thèse considère un cadre plus difficile mais plus général, dans lequel les noeuds se réveillent de façon asynchrone. Nous nous concentrons sur le problème de *désynchronisation à distance deux*, qui permet un contrôle de l'accès au support, primordial dans les réseaux sans fil. Nous montrons qu'il est possible pour les noeuds de communiquer de manière cohérente au-delà de leur voisinage immédiat. À cette fin, une primitive permettant aux noeuds de simuler une communication sur le carré du graphe de communication est présentée. Cette primitive est un élément central dans la conception de l'algorithme de désynchronisation à distance deux. Enfin, nous exploitons cette solution afin d'implémenter des primitives de haut niveau pour l'envoi et la réception de messages.

Title : Overcoming interference in the beeping communication model

Keywords : beeping model, interference control, wireless networks, weak devices, limited communication

Abstract : Small inexpensive inter-communicating electronic devices have become widely available. Although the individual device has severely limited capabilities (e.g., basic communication, constant-size memory or limited mobility), multitudes of such *weak devices* communicating together are able to form low-cost, easily deployable, yet highly performant networks. Such distributed systems present significant challenges however when it comes to the design of efficient, scalable and simple algorithms.

In this thesis, we are interested in studying such systems composed of *devices with severely limited communication* capabilities - using only simple bursts of energy. These distributed systems may be modeled using *the beeping model*, in which nodes communicate by beeping or listening to their neighbors (according to some undirected communication graph). Simultaneous communications (i.e., *collisions*) result in *non-destructive interference*: a node with two or more neighbors beeping simultaneously detects a beep. Its simple, general and energy efficient communication mechanism makes the beeping model widely applicable. However, that simplicity comes at a cost. Due to the poor expressiveness of beeps and the interference caused by simultaneous communications, algorithm design is challenging. Throughout the thesis, we overcome both difficulties in order to provide efficient communication primitives. A particular focus of the thesis is on deterministic and time-efficient solutions independent of the communication graph's parameters (i.e., uniform).

The first part of the thesis considers a setting in which nodes wake up at the same time (i.e., the network has been set up a priori). To obtain efficient solutions to *fundamental distributed communication problems*, we first focus on efficiently solving problems for *local* symmetry-breaking: $(\Delta+1)$-*vertex coloring* and *maximal independent set* (where Δ is the maximum degree of the communication graph). The solutions we devise are particularly efficient when the communication graph is sparse. They are then used to solve the 2-hop variants of these problems and to simulate message-passing. Finally, combining this simulation with existing results, which assume message-passing, gives the first vertex coloring algorithm using less than $\Delta + 1$ colors in the beeping model. Then, we study problems defined on a *global* scale, such as *leader election* and *multi-broadcast* (i.e., information dissemination). Leader election is a crucial building block in the design of distributed algorithms. We give the first two time-optimal leader election algorithms for the beeping model. One is deterministic, but requires unique identifiers. The second one does not need identifiers (useful for security and privacy reasons), but is randomized. Building upon the time-optimal leader election solution, computationally efficient and time-optimal algorithms for multi-broadcast are presented. Although a previous time-optimal solution was available, it required computationally expensive methods.

The second part of the thesis considers a more difficult but more general setting, in which nodes wake up at some arbitrary time rounds. We focus on the *desynchronization* problem, and more precisely on its 2-hop variant, which can be used as medium access control method. We show that it is possible for nodes to communicate in a coherent manner beyond their 1-hop neighborhood. More concretely, a primitive allowing nodes to simulate communication on the square of the communication graph is presented. This primitive is a centerpiece in the design of the 2-hop desynchronization algorithm. Finally, by leveraging this solution, we show that higher-level primitives for sending and receiving messages can be obtained in this difficult setting.

Université Paris-Saclay
Espace Technologique / Immeuble Discovery
Route de l'Orme aux Merisiers RD 128 / 91190 Saint-Aubin, France